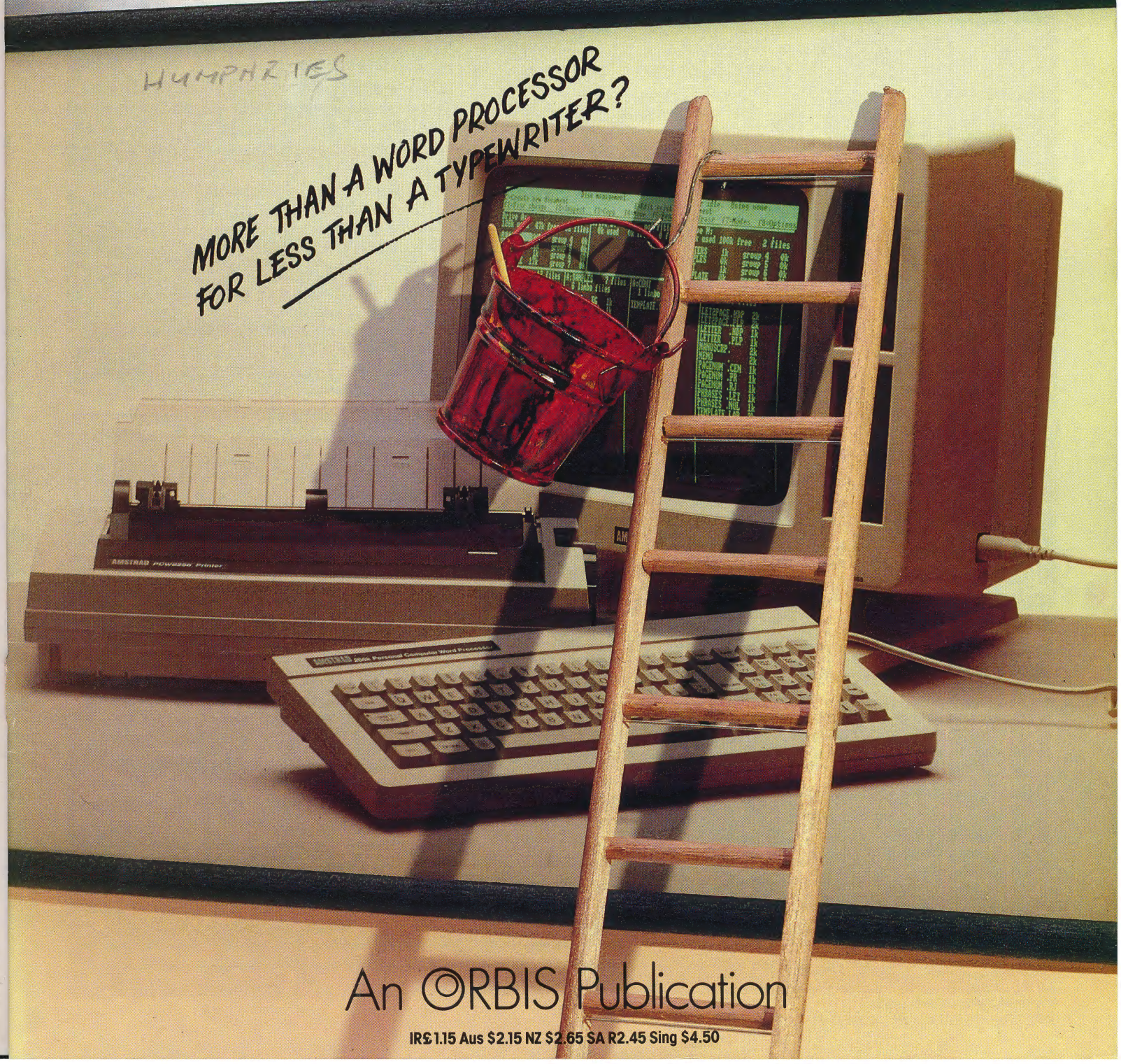
ISSN 0265-2919

90p 90

THE HOME COMPUTER ADVANCED COURSE

MAKING THE MOST OF YOUR MICRO



An ORBIS Publication

IR£1.15 Aus $2.15 NZ $2.65 SA R2.45 Sing $4.50

CONTENTS

APPLICATION

JOBS FOR THE BOYS? We look at some of the qualifications required for working with computers — 1781

HARDWARE

A DATE WITH JOYCE Amstrad's PCW 8256 is a complete word processing system — 1789

SOFTWARE

SMALL IS BEAUTIFUL We introduce a series on the Unix operating system with a discussion of its design philosophy — 1784

FOREST RANGER The use of hierarchical directories allows MS-DOS to handle a large number of files — 1792

FEED THE WORLD A chart-topping compilation cassette to raise money for the victims of famine in Africa — 1800

COMPUTER SCIENCE

CLASS CONSCIOUS Variables are recognised by the C compiler as belonging to one of four distinct classes — 1794

JARGON

FROM TRANSISTOR TO TREE A weekly glossary of computing terms — 1788

PROGRAMMING PROJECTS

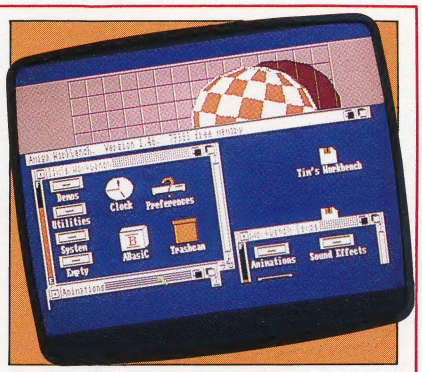
BANKER'S CARD This week we give the subroutines that deal with the computer's response to the player's hand — 1786

MACHINE CODE

COMMAND PERFORMANCE We look at the 68000's logical instructions, the shift and rotate operations, and program control — 1796

Next Week

- **The Commodore Amiga promises to revolutionise the design of microcomputers. We investigate the machine's development.**
- **Our series on Careers in Computing continues with a look at the opportunities for programmers and operators in the office and factory.**
- **We put the final touches to our pontoon programming project.**



QUIZ

1) What problems would confront you if you wished to use a different printer than that provided with the PCW 8256?
2) Name the four kinds of variables that are available in C?
3) What is TOPS?

Answers To Last Week's Quiz

1) A turnkey system is one in which a number of operating system commands are executed automatically on power up.
2) A program has to be 'installed' in order to take account of the particular non-standard expansion boards and peripherals that may be fitted to the computer.
3) Before accessing the Spectrum's video memory, the ULA examines whether the processor is using the memory, and if so stops the Z80's clock.

Coming Up

- Although few people use one, the VAX minicomputer is an integral part of the home micro industry. Many programs written for home micros are actually written on one of these machines, with the VAX emulating the performance of the specified microcomputer. We will be taking an in-depth look at the VAX in a future instalment.

FACT SHEET We conclude our series of fact sheets that detail the instruction set of the Motorola 68000 chip — INSIDE BACK COVER

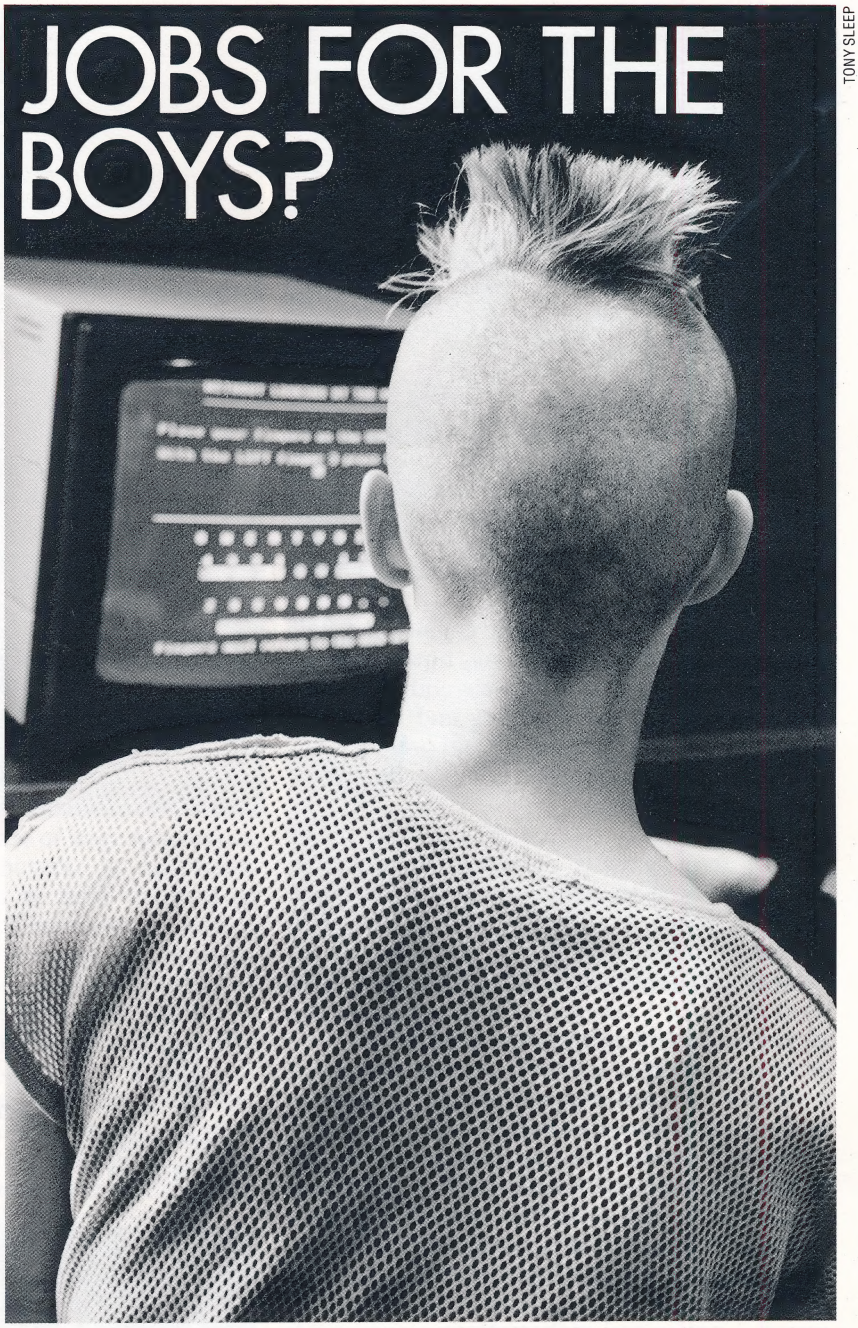
Editor Stephen Cooke; **Art Editor** Claudia Zeff; **Deputy Editor** Steve Colwill; **Production Editor** Bobby Pickering; **Designer** Julian Dorr; **Staff Writer** Steve Malone; **Art Assistant** Caroline Clayton; **Sub Editor** Jon Kaye; **Contributors** Mike Curtis, Steve Colwill, Steve Malone, Nigel Cross, David Fensome, Paul Walton, Peter Shaw; **Software Consultants** Pilot Software City; **Group Art Director** Perry Neville; **Managing Director** Stephen England; **Published by Orbis Publishing Ltd: Editorial Director** Brian Innes; **Project Development** Peter Brookesmith; **Executive Editor** Maurice Geller; **Production Assistant** Susan Brown; **Subscription Manager** Christine Allen; **Designed and produced by Bunch Partworks Ltd; Editorial Office** 14 Rathbone Place, London W1P 1DE;

© APSIF Copenhagen 1985; © Orbis Publishing Ltd 1985: Typeset by Universe; Reproduction by Mullis Morgan Ltd; Printed in Great Britain by Heanor Gate Printing Ltd, Derby

HOW TO OBTAIN ISSUES AND BINDERS FOR THE HOME COMPUTER ADVANCED COURSE – Issues can be obtained by placing an order with your newsagent or direct from our subscription department. If you have any difficulty obtaining any back issues from your newsagent, please write to us stating the issue(s) required and enclosing a cheque for the cover price of the issue(s). **AUSTRALIA** – please write to: Gordon & Gotch (Aus) Ltd, 114 William Street, PO Box 767G, Melbourne, Victoria 3001. **MALTA, NEW ZEALAND & SOUTH AFRICA –** Back numbers are available at cover price from your newsagent. In case of difficulty, write to the address given for binders.
UK/EIRE – Issue Price: 90p/IR£1.15. Subscription: 6 months: £26.00. 1 Year: £52.00. Binder: please send £3.95 per binder, or take advantage of our special offer in early issues. **EUROPE** – Issue Price: 90p. Subscription: 6 months air: £44.72. Surface: £36.14. 1 year air: £89.44. Surface: £72.28. Binder: £5.00. Airmail: £8.25. **MALTA** – Obtain binders from your newsagent or Miller (Malta) Ltd, MA Vassalli Street, Valetta, Malta. Price: £3.95. **MIDDLE EAST** – Issue Price: 90p. Subscription: 6 months air: £50.18. Surface: £36.14. 1 year air: £100.36. Surface: £72.28. Binder: £5.00. Airmail: £8.25. **AMERICAS/ASIA/AFRICA** – Issue Price: US/CAN$1.95/90p. Subscription: 6 months air: £59.54. Surface: £36.14. 1 year air: £119.08. Surface: £72.28. Binder: £5.00. Airmail: £9.50. **SOUTH AFRICA** – Issue Price: SA R2.45. Obtain binders from any branch of Central News Agency or Intermag, PO Box 57394, Springfield 2137. **SINGAPORE** – Issue Price: Sing $4.50. Obtain binders from MPH Distributors, 601 Sims Drive, 03-07-21, Singapore 1438. **AUSTRALASIA/FAR EAST** – Issue Price: 90p. Subscription: 6 months air: £64.22. Surface: £36.14. 1 year air £128.44. Surface: £72.28. Binder: £5.00. Airmail: £9.75. **AUSTRALIA** – Issue Price: AusS2.15. Obtain binders from First Post Pty Ltd, 23 Chandos Street, St Leonards, NSW 2065. **NEW ZEALAND** – Issue Price: NZS2.65. Obtain binders from your newsagent or Gordon & Gotch (NZ) Ltd, PO Box 1595, Wellington.
ADDRESS FOR BINDERS AND BACK ISSUES – Orbis Publishing Limited, Orbis House, Bedfordbury, London WC2 4BT. Telephone 01-379 5211. Cheques/postal orders should be made payable to Orbis Publishing Limited. Binder prices include postage and packing and prices are in stirling. Back issues are sold at the cover price, and we do not charge carriage in the UK.
NOTE – Binders and back issues are obtainable subject to availability of stocks. Whilst every attempt is made to keep the price of the issues and binders constant, the publishers reserve the right to increase the stated prices at any time when circumstances dictate. Binders depicted in this publication are those produced for the UK and Australian markets only. Binders and Issues may be subject to import duty and/or local taxes, which are not included in the above prices unless stated.
ADDRESS FOR SUBSCRIPTIONS – Orbis Publishing Limited, Hurst Farm, Baydon Road, Lambourn Woodlands, Newbury Berks, RG16 7TW. Telephone: 0488-72666. All cheques/postal orders should be made payable to Orbis Publishing Limited, Postage and packaging is included in subscription rates, and prices are given in sterling.

COVER PHOTOGRAPHY BY CHRIS STEVENS

JOBS FOR THE BOYS?



TONY SLEEP

Now that you have mastered the micro you may be thinking in terms of making a career in computing. Jobs are more plentiful in this industry than in most, offering higher than average pay and good prospects. This, the first in a series on choosing a career, provides an overview of working professionally with computers.

The first decision the prospective employee has to make is which part of this complex industry he wishes to work in, bearing in mind that good jobs are scarce all round. In this series we will consider careers that are as different from one another as chalk and cheese, each one offering a distinctive mixture of rewards: pay, prospects, technical involvement and job satisfaction.

One point to consider is that a career in computing can be a fixed path. If you start in one direction then that is the way you will continue for some time. This is true particularly if your employer invests heavily in training, or if you gain specialist knowledge and expertise.

Work varies enormously in computing, although there are some common job categories. Each job has different demands for staff, different rewards and different prospects for a career. Computer and electronics manufacturing, for example, requires three types of staff, all working on hardware, and the same is true of telecommunications, an increasingly popular area of the industry. There are line workers like those in any factory, with slightly better pay but the same prospects; production engineers, whose job is to ensure that the final products work; and research and design staff.

COMPUTER DESIGNERS

Computer design, whether of microprocessors or complete systems, is for the relatively experienced — or the truly inspired — but engineering is also open to those of more average talent. Both jobs require trained staff. Successful suppliers tend to employ more staff on developing the next product range.

These design jobs are the exclusive territory of graduates, who command high salaries, security and prestige. Graduates outnumber non-graduates in the industry by nine to one, according to COSIT, the Computing Services Industry Training Council. A recommended degree course to follow is one in electronic engineering or Computer Science at a university or polytechnic. Requiring O and A levels, the three-year course will teach you basic theory and advanced practice.

An alternative is a Higher National Diploma (HNC) or Ordinary National Diploma (ONC) from your local College of Further Education — this will satisfy many potential employers. A levels on their own are seldom enough: employers expect you to be able to find your way round a circuit board, in addition to knowing the principles on which a microprocessor operates.

Even the brightest programmers and electronics hobbyists will be unlikely to begin designing a PC or a chip. Other skills are essential before you will be allowed to design even a small part of either. A design project is usually shared among a team of a dozen or even several hundred workers, developing and completing sub-sections of the whole project.

Employers tend to believe that experience with academic work is an advantage in computing and electronics because it enables graduates to cope with the rate at which today's products are overtaken by new technologies. There is, however, an unfortunate belief that women are not able to accommodate change: the ratio of men to women in the industry is four to one (COSIT figures). The campaign for more Women in Science and Engineering (WISE) is fortunately doing much to

Brighter Prospects?
The computing industry offers a large number of openings to job seekers, but experience on home computers alone is unlikely to get you a job. For many people, a degree or — often better — a TOPS training offers a good start to a career which, although initially varied, will often involve specialisation in a particular field. In extreme cases, this can lead to narrowed employment prospects in later years unless a particular effort is made to keep pace with hardware and software innovation

Retail Sales Staff

An increasing number of young people start work in a computer store or a walk-in systems house through which small systems, peripherals, supplies and software are sold to the general public. Shop sales are infrequent (no more than half a dozen IBM PCs a week at an average branch of First Computer) and tend towards the low end of the market. In some cases a sale can develop into an account customer, but this is the exception.

The job allows the newcomer to learn about personal computers, but there is seldom time to become well enough acquainted with a wide range of computer systems or programming techniques. Nevertheless, for some people it is a valid entry point for a career in computer sales and marketing with a major manufacturer. Such jobs seldom pay well and the hours are long

change this view.

Enthusiasts can still become engineering staff with little or no formal training, however. Staff who can fix a piece of electronic equipment are always in demand, but training and pay (from £5,000 to £10,000) are limited and prospects are poor. Openings are often restricted to the smaller microcomputer firms.

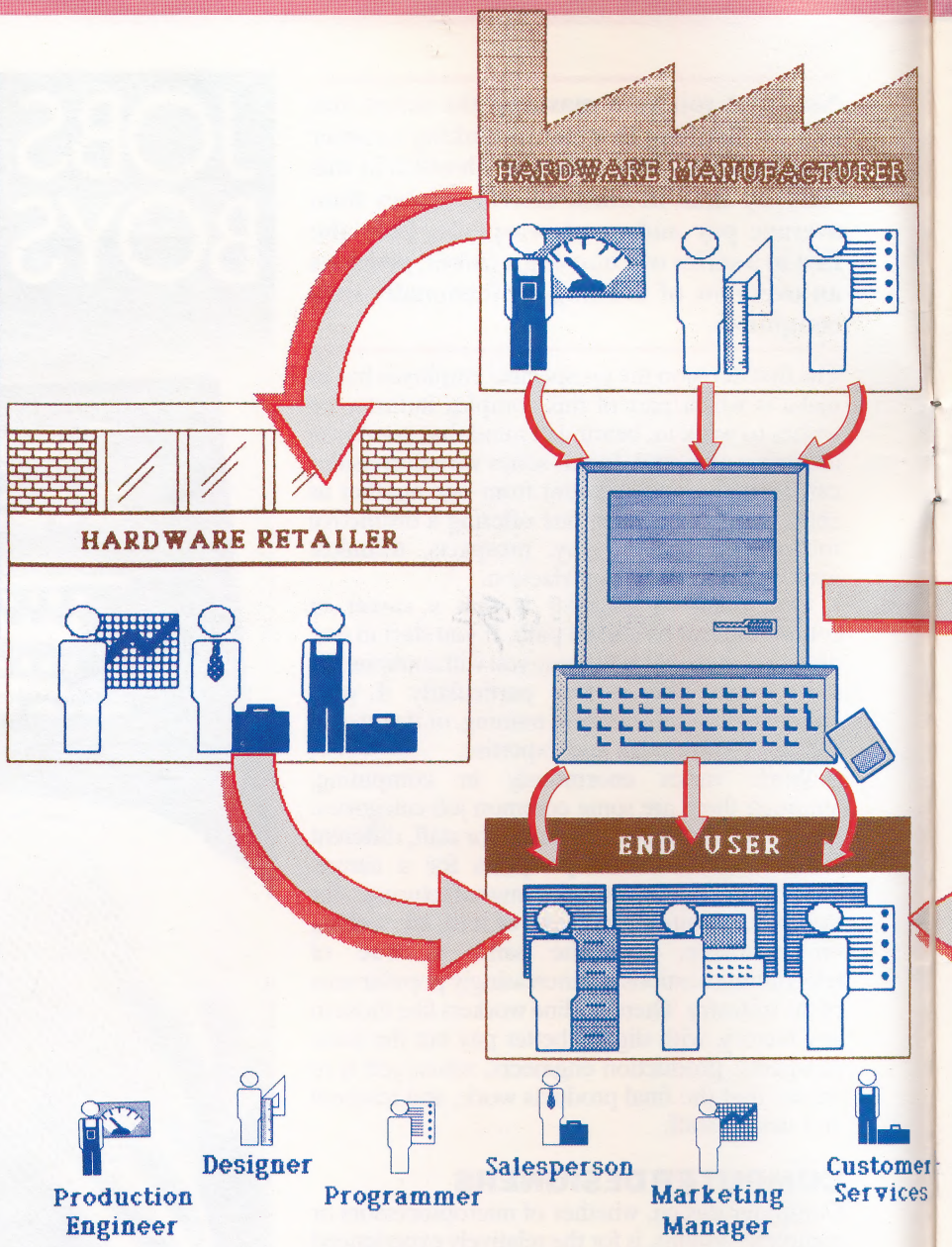
Software and services are the major source of jobs in the computer industry. Programmers (loosely defined to include other support staff) represent more than half of staff; sales and marketing staff, a third.

Most programers still work either in data processing (DP) operations — now developing into management information services (MIS) — or with computer suppliers, software houses or computer bureaux that sell computer services (normally time on a system).

DP is the most popular way to begin a career in computing, but it is diminishing in importance as computers become easier and more automatic to operate. Certain low-level job categories, such as operations, programming or basic systems analysis, can perhaps be viewed as a step in your career — not an end in themselves.

Programming itself is now a highly specified subject. An expert in a microcomputer language such as FORTH or PASCAL is unlikely to grasp the finer details of COBOL or FORTRAN on a mainframe immediately, and the situation is complicated by the predominance of PASCAL and COBOL in business, while FORTH and FORTRAN are more popular in technical areas. C is also gaining rapidly in importance, particularly where systems programming is concerned.

A significant proportion of people in the computer industry — perhaps a third of all staff — work for computer suppliers. Besides manufacturing, design and service support there are a great many salespeople. Sales staff can work for a computer supplier directly, for a distributor, for a systems or software house, or in one of the many retail outlets for microcomputers.



There's Gold In Them There Pay Cheques

Category	September 1984	March 1985
Computer manager	6.6	8.9
DP manager	9.6	8.6
Senior systems analyst	9.0	10.5
Senior analyst	7.3	7.9
Senior programmer	4.8	8.6
Analyst programmer	6.5	7.8
Programmer	5.7	1.6
All Executives	**6.6**	**7.6**

Pay Structure
The changing rate of pay for a range of computer personnel is shown in this table. The figures, representing some dramatic shifts in a little over six months, are taken from the Regional Reward Survey (September 1985). Figures for annual salary increases are averaged out across the country, and should be compared with inflation, which ran at around several per cent over the same period

In common with other sectors, the computer industry divides salespeople into three categories. First, there are 'direct' sales staff, although those 'on the road' for direct orders are a relative novelty. Most are given a number of accounts to manage.

Secondly, there are the marketing people, whose job is to generate leads with existing

Service Engineers

A service engineer is the closest thing to a simple repair man that the computer industry has, and he spends much time 'inside' the computer fixing faults. This involves a mixture of electrical and electronic engineering, and requires a basic knowledge of programming.

Pay and conditions are good — in compensation for the hardship of having to work across a wide geographical area and to be on call 24 hours a day. Factory-based engineers have a simpler job, finding and fixing faults in machines brought to them or in the process of manufacture.

Customer or field service engineers are recruited from a wide variety of related areas, excluding programming. The job requires above all great patience — finding a faulty chip in a room full of mainframe computers is taxing. Basic knowledge of programming helps the engineer to track a fault to a set of circuit boards or even to an individual board.

This type of job is becoming more scarce in the industry as computers become more reliable, and as on-line diagnostic systems take over the search for errors. In the near future the field service engineer may become simply the person who delivers a spare part, which the computer then locates

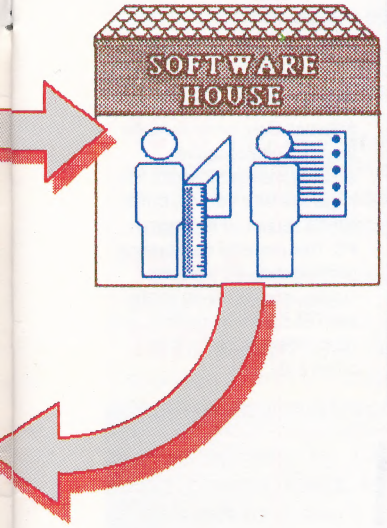
Computer Games Writing

The era of the teenage computer games millionaire is over — if it ever began. The notion that anybody could write a best-selling game which, like a pop record, would sell a million copies overnight was based on no more than a handful of questionable examples.

Games are written by a significant number of freelance programmers, many of whom began coming into the trade early in life with the advent of cheap and powerful home computers and a high demand for packaged software, including games.

Few programmers work in isolation. Most are commissioned as contract programmers by small firms or sell the rights to manufacture, market and distribute their programs to a company.

Such a freelance career presents significant problems. First you have to work every day, whether you feel like it or not, to make money. Many freelance programmers have a separate full-time job and work like this only part-time. Secondly, there has been an alarming failure rate among small software houses as the bottom fell out of the home computer and games markets. This makes choosing which product to market vitally important. To minimise the risk the business must be properly run. Finally, games writing is not, strictly speaking, a career. No-one is training you to program. There is a tendency to stagnate and return to what you do best and thus fail to develop new skills. In the long run this can damage both your business and your employment prospects



CAROLINE CLAYTON

Industrial Relations

Hardware remains at the hub of the computer industry, but the job seeker may well find more secure employment from specialising in one of the disciplines — marketing, for example — that revolve around it. Our diagram shows the various relationships between manufacturer, retailer, third-party manufacturer and the end-user

computer buyers (in the case of large systems) or, more rarely, first-time buyers. A mainframe computer company such as IBM trains these 'marketeers' to be the real sales people, talking directy to the the DP/MIS manager, the managing director or the board of the potential customer. A small microcomputer supplier will employ a marketing director to ensure that distributors and retailers stock its products.

The third sales job involves bringing together every element in the supply and support of hardware, software and systems. Management roles such as this in the computer industry are still often in the hands of ex-programmers or engineers.

Training is a maze of acronyms and it is essential, before starting a course, to find out what each set of initials will provide. The biggest supplier of training is the Manpower Services Commission (MSC), which runs courses mainly in programming and sofware techniques under the Training Opportunities Program (TOPS) scheme, designed largely for 're-training'.

The local Further Education (FE) colleges also run courses that cover much the same ground, with the addition of more hardware-oriented subjects, such as electronic engineering and the theoretical understanding of systems design. Such courses lead to the HNC or ONC, and programmers will expect to receive a City and Guilds (C&G) certificate.

This more academic approach is seen by employers in a less favourable light than the more practical TOPS courses. This system of training was last reviewed in 1983, when more than 3,000 people took TOPS and fewer than 1,000 gained other qualifications.

TOPS has been criticised as slow to update its courses to include modern methods, but with proposals from the Department of Trade and Industry Alvey committee on the table to right this flaw, it looks as though it will continue to be the major source of computer programming staff after the universities and polytechnics.

The Information Technology Education Centres (Itecs) are a third catchment scheme, with more than 150 centres set up since 1982 to train those young people caught in the trap between leaving school and getting a first job. Under the Youth Training Scheme (YTS), people between the ages 16 and 19 are paid to attend, and each Itec develops its own distinct brand of training in a cross between the FE college and the TOPS course.

The Itecs have been highly successful in training a new generation of programmers for positions in DP, engineers for computer manufacturers and sales staff for computer shops. Lack of funds, however, mean that many will close in the coming years unless the Government changes policy.

Computer manufacturers, meanwhile, are still among the highest paying and the best employers in terms of general conditions, training and future prospects. This is generally true in computer distributors and software houses, but quite the reverse in retail computer outlets.

SMALL IS BEAUTIFUL

We begin a series on the Unix operating system by looking at the philosophy behind it and the history of its development. After briefly considering the Unix shell, the part of the OS with which the user communicates, we see what happens when we first log in to the system.

Apart from basic input/output functions and file and memory management facilities, operating systems also provide a set of *tools* or *utilities* that create a proper working environment for the user. These utilities may range from compilers for a variety of languages to text editors and print formatters. Indeed, the main strengths of Unix, apart from the fact that it is a simple yet powerful and elegant operating system, are the large variety of tools that it provides and the way it enables output from one program to be used as the input to another. In doing so it permits a number of simple tools to be connected together to perform the most complicated tasks. The Unix philosophy can be simply stated in the following maxims:

1. Write small, simple programs that do one job really well.
2. Expect the output from any program to be used as the input to another, even if the second program has not yet been written.
3. Always write new programs to do new jobs, rather than modifying old ones; in that way you increase the library of tools available.

There is a snag, however. The idea of having a large variety of tools that can be interconnected to perform complex tasks is logical for programmers and other computer professionals, but it does not make the system very 'friendly' for inexperienced users. There is a great deal to remember if you want to make full use of a Unix system.

The only way to overcome this is to provide a system that inexperienced people can use: a user-friendly front end that hides the full power and complexity of Unix from the user. It is virtually impossible to do this, however, in such a way that the full power remains available, so Unix systems are found mainly in academic institutions rather than in business and home systems.

Unix was developed almost by accident by programmer Ken Thompson while working at Bell Laboratories, the research arm of the giant American AT&T Corporation. He was working on a simulation of planetary motion on a GE645 computer, which was running one of the first multi-user operating systems called Multics. The problems of developing software using an unfriendly environment on a large computer led him to transfer his efforts to a small DEC computer, a PDP-7. To do this effectively he wrote an operating system that gave most of the features of Multics on the smaller machine. It was so successful that it attracted the attention at Bell Laboratories of Dennis Ritchie and others, who by 1971 had developed the system into the first full working version of Unix.

Like most operating systems at the time, Unix was at first written in assembler, and ran properly only on the DEC minicomputer, mainly the PDP-11 series. However, Dennis Ritchie was also working on a language called B, a development of BCPL, the high-level language that gave almost the same direct control over the hardware as assembler. This soon developed into C and Unix was rewritten in this new language. Only a small fraction of Unix is still written in assembler, which makes it easy to transfer the code onto different machines — all you need is a C compiler. Today Unix runs on virtually every type of machine, from the largest mainframes down to 16-bit micros, and some restricted versions of Unix can even be run on eight-bit micros.

Until recently it was quite difficult and expensive to buy Unix for commercial use, but AT&T has a policy of giving almost free licences to academic institutions, so the use of Unix has established itself strongly in colleges and universities. The way has been led in the commercial world, particularly on

PC PAM
Although Unix is more usually found on minicomputer systems, Hewlett-Packard has implemented a version, called HP-UX, to run on its Integral PC. This powerful multitasking portable includes a plasma screen, integral ink-jet printer and 768 Kbytes of memory, and runs under a WIMP system called PAM

LIZ HEANEY

The Unix Story

The advent of the Unix operating system attracted many computer users dissatisfied with pre-Unix computer environments. Because Unix was developed by a small team of people it has remained sensible and compact. Many of its excellent facilities have found their ways into relatively new operating systems such as MS-DOS

1969 Dennis M. Ritchie and Ken Thompson, working at Bell Laboratories in the USA, develop an operating system for their own convenience to run on a PDP-7 computer. They christen it 'Unix' as a pun on the large Multics operating system current at the time

1971 Unix is transferred to PDP-11 computers

1973 The original versions of Unix were written in assembly language. Nearly the whole of Unix is recoded in C, making it easily portable to other computers, which require only a C compiler

1974-5 PWB/Unix is introduced — essentially a large version of Unix

1977 Version 7 Unix developed to rid the operating system of machine-dependent features. This is the version of Unix currently used

microcomputers, by Unix 'lookalikes' such as Xenix, Cromix, Onyx, Idris, Coherent and OS-9, all of which have many features in common with Unix. Now, operating systems like CP/M and MS-DOS are acquiring Unix-like features.

Pure Unix may never be a major commercial success but it has had a profound effect on operating system development and will continue to be important for a long time. The current versions include the Bell Laboratories version 7, soon to be replaced by version 5, and the offshoot developed at the University of California at Berkeley which has reached version 4.2. Examples cited here are all taken from the Berkeley 4.2 version, but they should apply equally well to most others.

THE SHELL

A major feature of Unix is the shell, the part of the OS with which the user communicates. It is much more than the simple command processor found in many operating systems, for it can be customised, by both the system provider and the individual user, to incorporate new commands and a different user interface if required. It also provides many of the features of a programming language such as C, and in conjunction with the wide variety of tools it is possible to write large and complex programs without using a programming language at all.

Initially, we will look at some of the standard commands, and then at the ways in which we create commands and new names for existing commands. The format for most Unix commands is the same:

```
command_name  options  file_or_directory_name
```

where each option begins with a minus sign. An important fact to remember about Unix is that it is case-sensitive — that is, the names 'FRED', 'Fred' and 'fred' are all considered different. It is a common mistake to use capital letters inadvertently for a command and find that it does not work (most standard Unix commands use lower case letters).

The first stage in using a Unix system (or any other multi-user system for that matter) is to *log in.* This is done automatically whenever you connect to the system, but it may be done at any time thereafter by typing the command login. The system will give you the prompt:

```
login:
```

At which point you must type your unique user identification name. The system responds with:

```
password:
```

at which point you type your own password, which is not echoed on the screen. Passwords are normally at least six characters long. If you make a mistake in entering your password it has to be repeated, but to avoid unauthorised people finding out passwords by trial and error the system may refuse to let you try again after a number of mistakes. Once the procedure has been completed the system will usually type a welcome message and eventually the operating system prompt.

At this stage Unix will run your own customising shell program contained in a special file, .login, which among other things may define your own personal prompt. The standard Unix prompt is %. You will now have been recognised as an authorised user and Unix will automatically have connected you to your own area on the disk where you keep all your files. It is possible to access files on other areas, particularly those designated as public, but if required you can protect any files and areas from unauthorised access. If you wish to change your password, which it is a good idea to do regularly, you may do so by typing the command passwd, which will ask you to type first your old password and then your new password twice.

The best way to find your way around a new computer system is to try it out and Unix provides two useful facilities for helping you to get to know the system. Unix is almost unique in operating systems in that it includes an on-line manual with entries for virtually every command or topic you might wish to use. The command that allows you to see a manual entry is:

```
man topic_name
```

For novice users, Unix incorporates a self-instructional program that is initiated by the command learn, which will give lessons on a variety of topics.

To end a session with Unix the command logout should be issued; you will be disconnected from the system until you next login.

BANKER'S CARD

We are now at the stage in our pontoon project where the player can play out a complete hand. In this instalment we look at the routines that allow the banker (whose role is taken by the computer) to respond intelligently in countering the player's hand.

Once the player has completed his hand it is the banker's turn. The banker has several advantages in the version of the rules of pontoon we have adopted for our program. First, the banker knows what score the players has achieved and so does not have to make risky twists that might bust the hand. The second advantage is that the banker has only to draw with you to win the round and take your money. To offset somewhat these advantages the program does not allow the banker to burn a hand of 12, 13 or 14.

At this stage of the game the screen will display your cards and the two cards that were initially dealt to the bank. The first of these was dealt face-down, so the next task of the program is to turn the card over. The simplest way of doing this is to erase the banker's cards, using the general erase routine at line 670, and then reprint them face up by setting CN and SU (the card and suit numbers) to the values held in the banker's section of the hand array, HD(,,), and finally call the card display routine we developed earlier in the series (see pages 1712 and 1729). This is all done by lines 150 to 170 of the main program loop. The program is now in a position to play out the banker's hand automatically.

The Computer's Turn

Amstrad CPC Range

```
130 REM **** computer's turn ****
140 pl=2
150 ep=20:GOSUB 670:REM erase cards
160 cn=hd(pl,1,1):su=hd(pl,1,2):GOSUB 10
00:REM reprint
170 cn=hd(pl,2,1):su=hd(pl,2,2):GOSUB 10
00:REM reprint
180 GOSUB 2500:REM bank twist etc
190 REM **** win or lose ****
200 IF bw=1 THEN GOSUB 700:PEN black:PRI
NT"sorry you lose":GOTO 300
210 GOSUB 700:PEN black:PRINT"you win "
300 a$="":WHILE a$="":a$=INKEY$:WEND

305 REM ** if shift/s then shuffle **
310 IF a$="S" THEN GOSUB 700:PRINT"shuff
ling....please wait":GOSUB 3000
320 GOTO 50
2500 REM **** bank's turn ****
2520 ON pv+1 GOSUB 2540,2550,2560,2570,2
580
2530 RETURN
2540 bw=1:RETURN:REM punter bust
2550 ts=ps:GOSUB 5000:RETURN:REM twist u
ntil beat punter or bust
2560 ts=21:GOSUB 5000:RETURN:REM twist u
ntil pontoon or bust
2570 GOSUB 5200:RETURN:REM twist until 5
 card trick or bust
2580 GOSUB 800:IF ef=2 THEN bw=1:RETURN:
REM bank's royal pontoon
2585 bw=0:RETURN:REM punter's royal pont
oon
5000 REM **** bank twist until target or
 bust ****
5010 GOSUB 800:REM evaluate
5020 IF ef=4 THEN bw=0:RETURN:REM bust
5025 IF ef=2 OR ef=5 THEN bw=1:RETURN:RE
M bank royal pontoon/five card trick
5040 IF tt(2,1)>=ts OR(tt(2,2)<=21 AND t
t(2,2)>=ts) THEN bw=1:RETURN
5045 IF hp(2)>5 THEN bw=0:RETURN:REM fiv
 cards dealt
5050 GOSUB 1300:GOTO 5000:REM deal and r
e-evaluate
5200 REM **** twist until 5 card trick o
r bust ****
5220 GOSUB 800:REM evaluate
5225 IF ef=4 THEN bw=0:RETURN:REM bust
5227 IF ef=2 OR ef=5 THEN bw=1:RETURN:RE
M bank royal pontoon/five card trick
5228 IF hp(2)>5 THEN bw=0:RETURN:REM fiv
 cards dealt
5230 GOSUB 1300:GOTO 5200:REM deal card
and eval again
```

Sinclair Spectrum

```
 130>REM **** COMPUTER'S TURN ****
 140 LET PL=2
 150 LET EP=14: GO SUB 670: REM ERASE CA
RDS
 160 LET CN=D(PL,1,1): LET SU=D(PL,1,2):
 GO SUB 1000: REM REPRINT
 170 LET CN=D(PL,2,1): LET SU=D(PL,2,2):
 GO SUB 1000: REM REPRINT
 180 GO SUB 2500: REM BANK TWIST ETC
 190 REM **** WIN OR LOSE ****
 200 IF BW=1 THEN  GO SUB 700: PRINT "SO
RRY YOU LOSE": GO TO 300
 210 GO SUB 700: PRINT "YOU WIN #";BT
 300 LET A$=INKEY$: IF A$="" THEN  GO TO
 300
 305 REM ** IF SYM/S THEN SHUFFLE **
 310 IF A$=CHR$ 195 THEN  GO SUB 700: PR
INT "SHUFFLING ... PLEASE WAIT": GO SUB
3000
 320 GO TO 50
2500>REM **** BANK'S TURN ****
2520 GO SUB (PV*10)+2540
2530 RETURN
2540 LET BW=1: RETURN : REM PUNTER BUST
2550 LET TS=PS: GO SUB 5000: RETURN : RE
M TWIST UNTIL BEAT PUNTER OR BUST
2560 LET TS=21: GO SUB 5000: RETURN : RE
M TWIST UNTIL PONTOON OR BUST
2570 GO SUB 5200: RETURN : REM TWIST UNT
ILFIVE CARD TRICK OR BUST
2580 GO SUB 800: IF EF=2 THEN  LET BW=1:
 RETURN : REM ROYAL PONTOON
2585 LET BW=0: RETURN : REM PUNTER'S ROY
AL PONTOON
5000>REM **** BANK TWIST UNTIL TARGET OR
 BUST
5010 GO SUB 800: REM EVALUATE
5020 IF EF=4 THEN  LET BW=0: RETURN : RE
M BUST
5025  IF EF=2  OR EF=5 THEN  LET BW=1: R
ETURN
5040 IF T(2,1)>=TS OR (T(2,2)<=21 AND T(
2,2)>=TS) THEN  LET BW=1: RETURN
5050 GO SUB 1300: GO TO 5000: REM DEAL A
ND RE-EVALUATE
5200 REM **** TWIST UNTIL 5 CARD OR BUST
 ****
5220 GO SUB 800: REM EVALUATE
5225 IF EF=4 THEN  LET BW=0: RETURN : RE
M BUST
5227  IF EF=2  OR EF=5 THEN  LET BW=1: R
ETURN
5228  IF P(2)>5  THEN  LET BW=0: RETURN
5230 GO SUB 1300: GO TO 5200
```

If the player is bust, no further action need be taken. The banker has won and there is obviously no need to ask for extra cards.

If the player has a score of less than 21, the banker must twist repeatedly until that score is equalled, beaten or the banker busts.

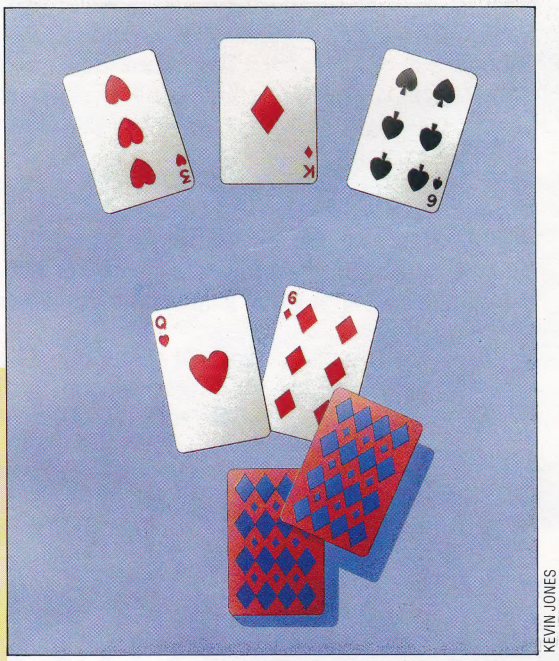
If the player has pontoon, the banker must twist until getting 21 or going bust.

If the player has a five-card trick then, because a five-card trick beats ordinary pontoon, the banker must twist in an attempt to obtain a five-card, but may of course bust in the attempt.

If the player has royal pontoon, the banker can win only by obtaining royal pontoon with the two cards already dealt.

All these actions are, of course, unnecessary if the banker has royal pontoon — a state that cannot be beaten.

The subroutine at line 2500 selects the course of action, using the ON . . . GOSUB command in conjunction with the variable PV. In the case of twisting to beat the player's score, a target score TS is set and a further subroutine at line 5000 is called. If the banker has to twist to obtain a five-card trick, a slightly different routine is needed and this is located at line 5200. In all cases the banker's turn will terminate in the bank winning or losing, indicated by a flag BW being set to 1 or 0 accordingly. The value of this flag determines which end-of-game message is printed.



KEVIN JONES

Bank Statement
In this version of pontoon, the banker has the advantage of knowing what hand needs to be beaten. In this example game, the punter has stuck on 19. The banker knows that he must deal himself at least one more card in an attempt to win the game

Commodore 64

```
130 REM **** COMPUTER'S TURN ****
140 PL=2
150 EP=20:GOSUB670:REM ERASE CARDS
160 CN=HD(PL,1,1):SU=HD(PL,1,2):GOSUB100
0:REM REPRINT
170 CN=HD(PL,2,1):SU=HD(PL,2,2):GOSUB100
0:REM REPRINT
180 GOSUB2500:REM BANK TWIST ETC
190 REM **** WIN OR LOSE ****
200 IF BW=1 THEN GOSUB700:PRINTCHR$(156)
;"SORRY YOU LOSE":GOTO 300
210 GOSUB700:PRINT CHR$(156);"YOU WIN"
300 GET A$:IF A$=""THEN 300
305 REM ** IF SHIFT/S THEN SHUFFLE **
310 IF A$=CHR$(211)THENGOSUB700:PRINT"SH
UFFLING ... PLEASE WAIT":GOSUB 3000
320 GOTO 50
2500 REM **** BANK'S TURN ****
2520 ON PV+1 GOSUB 2540,2550,2560,2570,2
580
2530 RETURN
2540 BW=1:RETURN:REM PUNTER BUST
2550 TS=PS:GOSUB5000:RETURN:REM TWIST UN
TIL BEAT PUNTER OR BUST
2560 TS=21:GOSUB5000:RETURN:REM TWIST UN
TIL PONTOON OR BUST
2570 GOSUB5200:RETURN:REM TWIST UNTIL 5
CARD TRICK OR BUST
2580 GOSUB800:IF EF=2 THEN BW=1:RETURN:R
EM ROYAL PONTOON
2585 BW=0:RETURN:REM PUNTER'S ROYAL PONT
OON
5000 REM **** BANK TWIST UNTIL TARGET OR
 BUST ****
5010 GOSUB800:REM EVALUATE
5020 IF EF=4 THEN BW=0:RETURN:REM BUST
5025 IF EF=2 OR EF=5 THEN BW=1:RETURN
5040 IF TT(2,1)>=TS OR(TT(2,2)<=21AND TT
(2,2)>=TS)THEN BW=1:RETURN
5045 IF HP(2)>5 THEN BW=0:RETURN:REM FIV
E CARDS DEALT
5050 GOSUB 1300:GOTO 5000:REM DEAL AND R
E-EVALUATE
5200 REM**** TWIST UNTIL 5 CARD OR BUST
****
5220 GOSUB800:REM EVALUATE
5225 IF EF=4 THEN BW=0:RETURN:REM BUST
5227 IF EF=2 OR EF=5 THEN BW=1:RETURN
5228 IF HP(2)>5 THEN BW=0:RETURN
5230 GOSUB 1300:GOTO 5200:REM DEAL CARD
```

BBC Micro

```
  130 REM
  140 PL=2
  150 EP=20:GOSUB 670
  160 CN=HD(PL,1,1):SU=HD(PL,1,2):GOSUB
1000
  170 CN=HD(PL,2,1):SU=HD(PL,2,2):GOSUB
1000
  180 GOSUB 2500
  190 REM
  200 IF BW=1 THEN GOSUB 700:PRINT"SORRY
, YOU LOSE!":GOTO 300
  210 GOSUB 700:PRINT"YOU WIN "
  300 A$=GET$
  305 REM
  310 IF A$="S" THEN GOSUB 700:PRINT"SHU
FFLING....PLEASE WAIT":GOSUB 3000
  320 GOTO 50
 2500 REM
 2520 ON PV+1 GOSUB 2540,2550,2560,2570,
2580
 2530 RETURN
 2540 BW=1:RETURN
 2550 TS=PS:GOSUB 5000:RETURN
 2560 TS=21:GOSUB 5000:RETURN
 2570 GOSUB 5200:RETURN
 2580 GOSUB 800:IF EF=2 THEN BW=1
 2585 BW=0:RETURN
 5000 FORDL=1TO100:NEXT
 5010 GOSUB 800
 5020 IF EF=4 THEN BW=0:RETURN
 5025 IF EF=2 THEN BW=1:RETURN
 5040 IF TT(2,1)>=TS OR (TT(2,2)<=21 AND
 TT(2,2)>=TS) THEN BW=1:RETURN
 5050 GOSUB 1300:GOTO 5000
 5200 REM
 5210 GOSUB 1300
 5220 GOSUB 800
 5225 IF EF=4 THEN BW=0
 5230 IF EF<>5 THEN 5210
 5240 BW=1:RETURN
```

TRANSISTOR

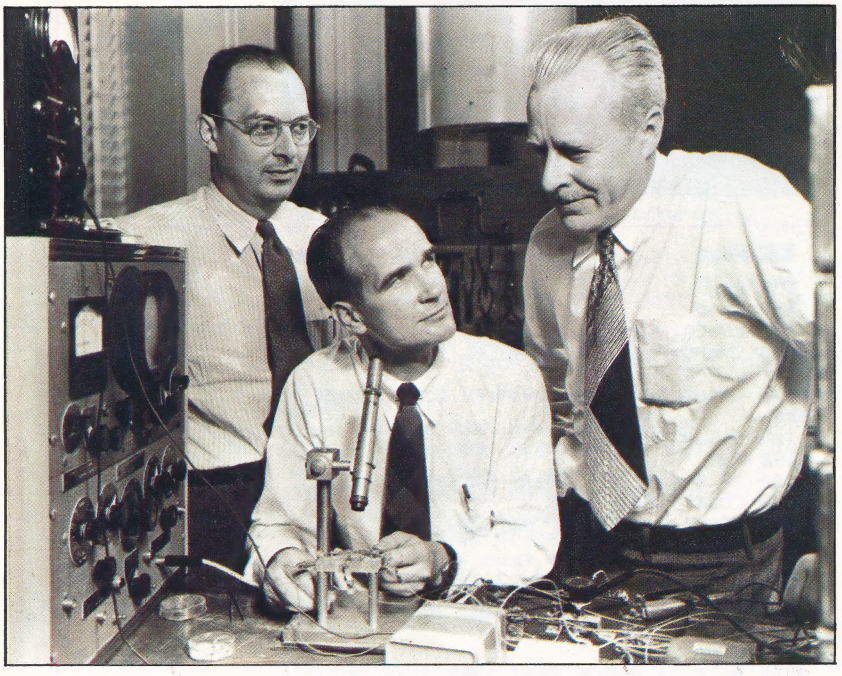
The *transistor* is a semiconductor device consisting of three terminals, known as the base, collector and emitter. When a current is applied to the base of a transistor it reduces the resistance within the semiconductor and enables a current to flow between the collector and emitter. The greater the current to the base the larger the current that flows to the emitter. From this effect both the major uses of the transistor follow. The first is its role as a voltage or current amplifier. Secondly, as a current can flow from the emitter only if there is a current present at both the base and the collector, the transistor can be viewed as a logical switching element.

Most transistors are manufactured from pure silicon, which is then doped — another element is added — to give the device its semiconductor properties. However, some transistors are now being manufactured using gallium, which allows faster switching.

There are two basic types of transistor. The bipolar transistor consists of slices of silicon, each of which has been doped with a different element. This type is designed so that the emitter will not produce a current until both the base and collector are at their maximum voltages. Thus only a very small change in the voltage of either of the two input currents is required to switch the output current on or off. This is a fast and efficient way of producing a digital signal and bipolar transistors are used exclusively on fast mainframe computers and in certain registers in microcomputers.

Bipolar transistors are, however, expensive to produce, and in microcomputers where switching speed is not essential the cheaper Field Effect Transistor (FET) is used for such things as memory storage. In this type the three terminals are called the source, drain and gate. The flow of current through the device passes through an area of the transistor known as the 'channel'. The amount of current that flows through the channel is determined by the charges on either side of the channel from the source and gate. Increasing the potential difference (the strength of the field) between the source and gate will increase the current flow through the channel, which emerges at the drain.

Physical Scientists
The 1956 Nobel Prize for Physics was awarded to the team of American scientists whose research led to the invention of the transistor. Pictured here at the Bell Telephone Laboratories, where they conducted their research, are (left to right) John Bardeen, William Shockley and Walter Brattain



TRANSLATION TABLE

A *translation table* is held within the memory of a computer, or sometimes a peripheral, and is used to convert codes from one format to another. For example, a word processor may have a number of Escape codes that perform certain text formatting functions, while the printer connected to it may have a different set of codes to perform the same functions. In this situation a translation table would need to be set up within the computer to enable the codes to be translated from the word processing format to those of the printer to produce the correct results.

TRANSMISSION RATE

The *transmission rate* refers to the speed at which data is transmitted through a circuit or between devices. Transmission rates between computers are usually measured in bits per second, known as the baud rate. For example, most cassette loading and saving systems transmit at 300 baud (300 bits per second).

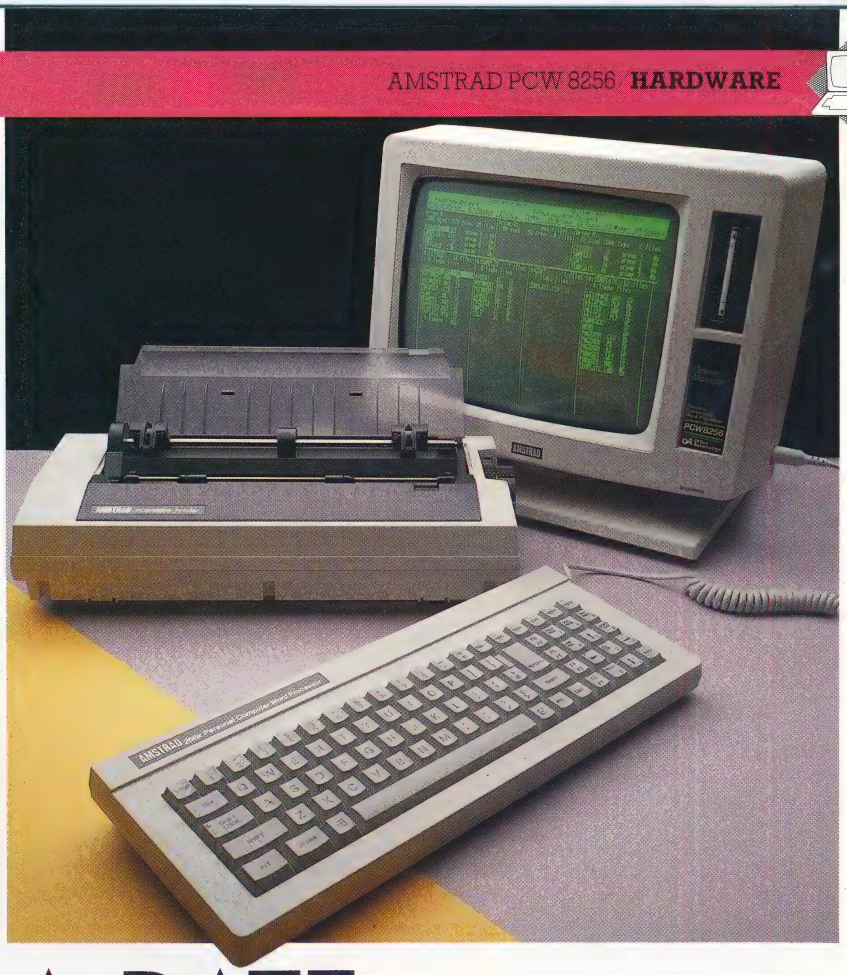
Although some protocols require a standard transmission rate many computers are capable of transmitting only at one speed. An interface is therefore required that will upgrade or downgrade the transmission rate to the appropriate speed.

TREE

A *tree* is a data structure formed from a number of pathways that connect a number of nodes together. Although it is possible to have a single pathway leading from a node, this is not common as nodes are generally used as points where a decision has to be made as to which direction to follow. The process whereby the computer responds to the decisions and follows a particular path is referred to as a 'tree search'.

Typically a tree is represented as a single node at the top, known as the 'root'. From the root, the tree branches downwards, increasing the number of nodes at each subsequent level. The nodes at the lowest levels are often referred to as 'leaves', and tree structures often contain other, smaller trees, known as 'sub-trees'.

Many different types of tree exist. A common example is the binary tree, which produces two branches from each node and is usually used to represent a knowledge structure based on yes/no decisions. Generally, tree data structures are useful where data needs to be related or ordered in a hierarchy. Trees are commonly used in databases where a key or series of keys dictate the path that will obtain the required information.



A DATE WITH JOYCE

An integrated word processing system with an entry price of £400, including monitor, printer and disk drive, is excellent value for money. But concern about the quality of the printer and the bundled word processing software may darken what could be a very bright future for Amstrad's PCW 8256.

Amstrad Consumer Electronics has a history of packaging existing technology into all-in-one systems. The Amstrad PCW 8256, though a very different machine from its predecessors, has been built in much the same vein. Designed as an integrated word processor package, the system includes in its listed price a built-in disk drive, dot matrix printer and the word processing software.

Even taking into account the company's manufacturing and marketing philosophy, the launch of the PCW 8256, nicknamed 'Joyce' after Alan Sugar's secretary, still caused a surprise in the industry. Whereas previous Amstrad computers were designed to appeal to the widest possible market, the latest machine is the company's first to be aimed at a specific sector, albeit a large one. So while the Amstrad PCW 8256 can be used as a standard computer, the company is aiming it squarely at small businesses.

As with previous Amstrad computers, the monitor is housed in a case with the power supply alongside a built-in disk drive and a socket that provides power to the stand-alone printer. A coiled cable from the keyboard plugs into a socket on the side.

The tightly packed 82 keys include the standard QWERTY group plus some additional ones such as Alt and Extra to provide alternative character sets. These include Greek characters and others, such as ü, enabling the computer to process a variety of languages. To the immediate right of these are four programmable function keys, which have a number of uses, providing up to eight different functions. A cursor cluster nests in the bottom right-hand corner of the keyboard and above that are several keys dedicated to various word processing functions. Although an improvement on the standard Amstrad keyboard, the keys rattle somewhat when text is input with any speed.

GREEN SCREEN MONITOR

The monitor is a standard 'green screen' model, which Amstrad says will be the only type available. With a resolution of 92 by 32 characters, the screen is larger than those provided on most computers. On screen, the characters are of the same fount, but larger than on other Amstrad models, making the text easier to read. The disk drive is in the top right-hand corner of the monitor casing, below which is another slot for a second disk drive to be fitted, behind the name plate.

The five by eight pin dot matrix printer provided with the PCW 8256 is capable of accommodating single A4 sheets or the standard 11in continuous feed. Of the two modes in which it can operate, the first is known as 'draft mode' and, as its name suggests, is ideal for producing draft copies, simple memos and so on. This mode, however, clearly illustrates the printer's limitations. Close inspection of the print shows that the individual dots are not capable of forming a continuous line.

Amstrad doesn't manufacture its own printer, but 'badge-engineers' the model from other companies, a policy that has elicited some complaints from users. What's more important, however, is that a good-quality printer is essential to a word processing system, so you would be justified in questioning Amstrad's decision to use this particular model with the PCW 8256. The company apparently recognised this and announced that an RS232 and Centronics interface would be made available, enabling higher-quality printers to be fitted to the basic machine.

You will be confronted with several problems, however, if you decide to attach another printer. Because LocoScript, the word processing software bundled with the machine, is very much based around the existing hardware, it has difficulty communicating with other printers. Amstrad's solution is to modify the CP/M 3.0 operating

Amstrad Means Business
The PCW 8256, nicknamed Joyce, is Amstrad's first micro aimed mainly at the business market. Although it is being marketed as a dedicated word processor — with monitor, printer, disk drive and bundled word processing software — the PCW 8256 is also a very powerful eight-bit computer

system to be able to read the files in ASCII format and send them to the printer. This means, of course, that you will have to quit LocoScript in order to print the file. Furthermore, many of the more attractive print formatting features of LocoScript cannot be included in such a system.

The 'high-quality' print mode nevertheless overcomes these hardware limitations. The printer goes over each line twice, filling in the gaps left by the dots, in much the same way as bold face type is achieved. Although this greatly improves the quality of print, it drastically reduces the speed of the printer, from 90 characters per second (cps) in draft mode down to 20 cps. There are some doubts concerning the reliability of the printer itself. The centre row of pins in the review system failed after only two days of use, which resulted in gaps in the printed characters.

Like the previous Amstrad computers, the PCW 8256 is based around the familiar Z80 processor fitted with 256 Kbytes of RAM. The now common 'bank switching' techniques are incorporated into the machine, which would otherwise be incapable of handling so much memory. Around 110 Kbytes have been organised as a silicon disk, which the processor treats as a floppy disk and so accesses information accordingly. Since the information is actually held on RAM chips, it can be accessed almost instantaneously, which, of course, is much faster than from a disk.

BUNDLED SOFTWARE

The software bundled with the PCW 8256 includes CP/M 3.0, an improved version of the operating system included with earlier Amstrad machines (see page 1629). Amstrad's AMSDOS disk operating system is also included, along with Digital Research's GSX graphics package. This enables CP/M to display graphics (a facility that was not originally included in its design) and allows a number of business packages available under CP/M to make use of the graphics feature. Amstrad has also bundled with the machine Mallard BASIC, Dr LOGO and LocoScript. Amstrad claims that LocoScript has been designed to be powerful and yet as easy to use as an electric typewriter.

On power up, the LocoScript screen is dotted with TAB markers, which are intended to assist in the formatting of the text. However, many users might feel that this produces a rather messy display, which merely gets in the way of the text. The programmers have made it possible, though, for the user to remove all these markers, which makes for a much cleaner display.

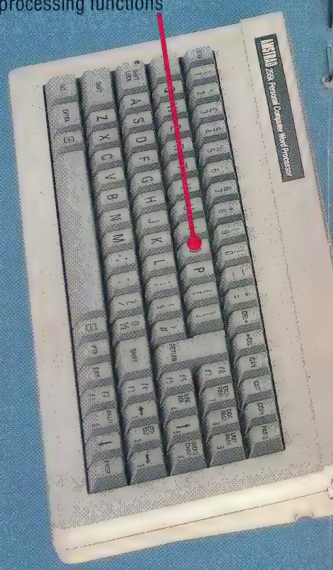
The system certainly appears to be as powerful and flexible as many word processing systems designed for high-quality business machines. LocoScript has facilities to allow text to be moved around within a document, move the cursor by character, word or paragraph, and to locate words and phrases — all tasks that are performed by the specialised word processing keys fitted to the keyboard. To copy an area of text from one part of a document to another, for example, requires moving the cursor to the beginning of the area in question and pressing the Copy key. The process is then repeated at the end of that section. The transfer is performed by moving the cursor to the relevant position and pressing the Paste key.

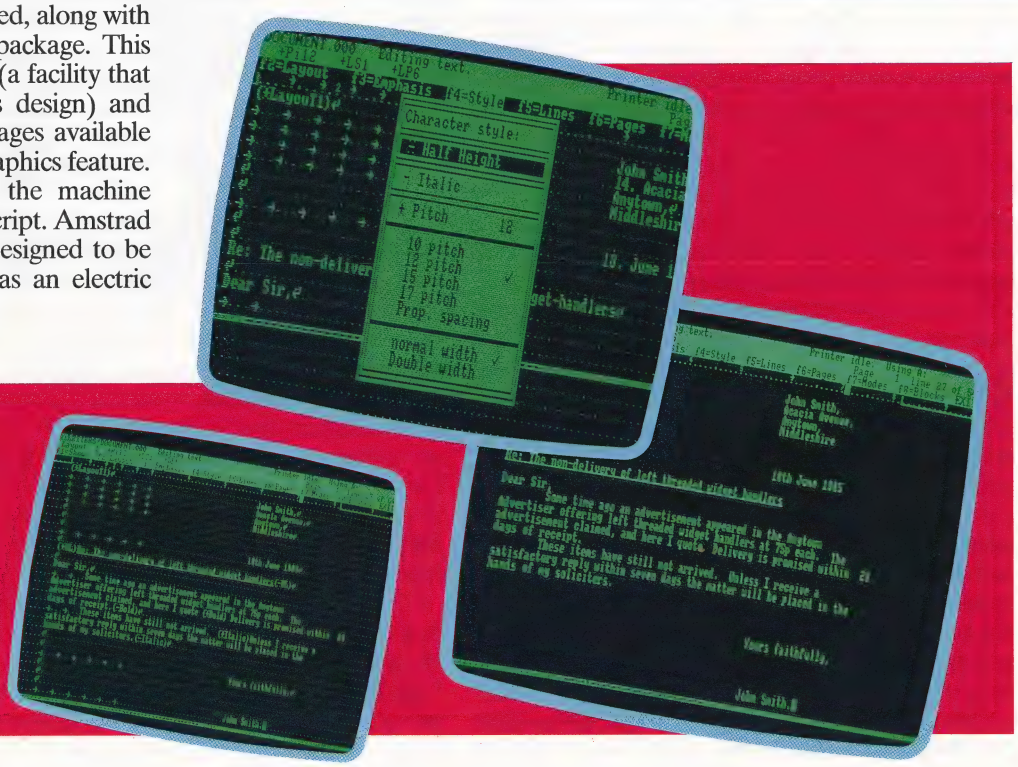
Facilities such as justification, character emphasis and print formats are accessed via the function keys. A pull-down menu will appear with a series of options that can be selected by using the cursor keys and then pressing the Enter key. This system of menus and submenus has been designed to augment the word processing facilities.

As we've seen in our series on word processing packages, most make extensive use of Control characters and separate screen menus to carry out the wide variety of functions available. It's understandable why the programmers of the PCW 8256 have chosen to use this system of programmable function keys, menus and

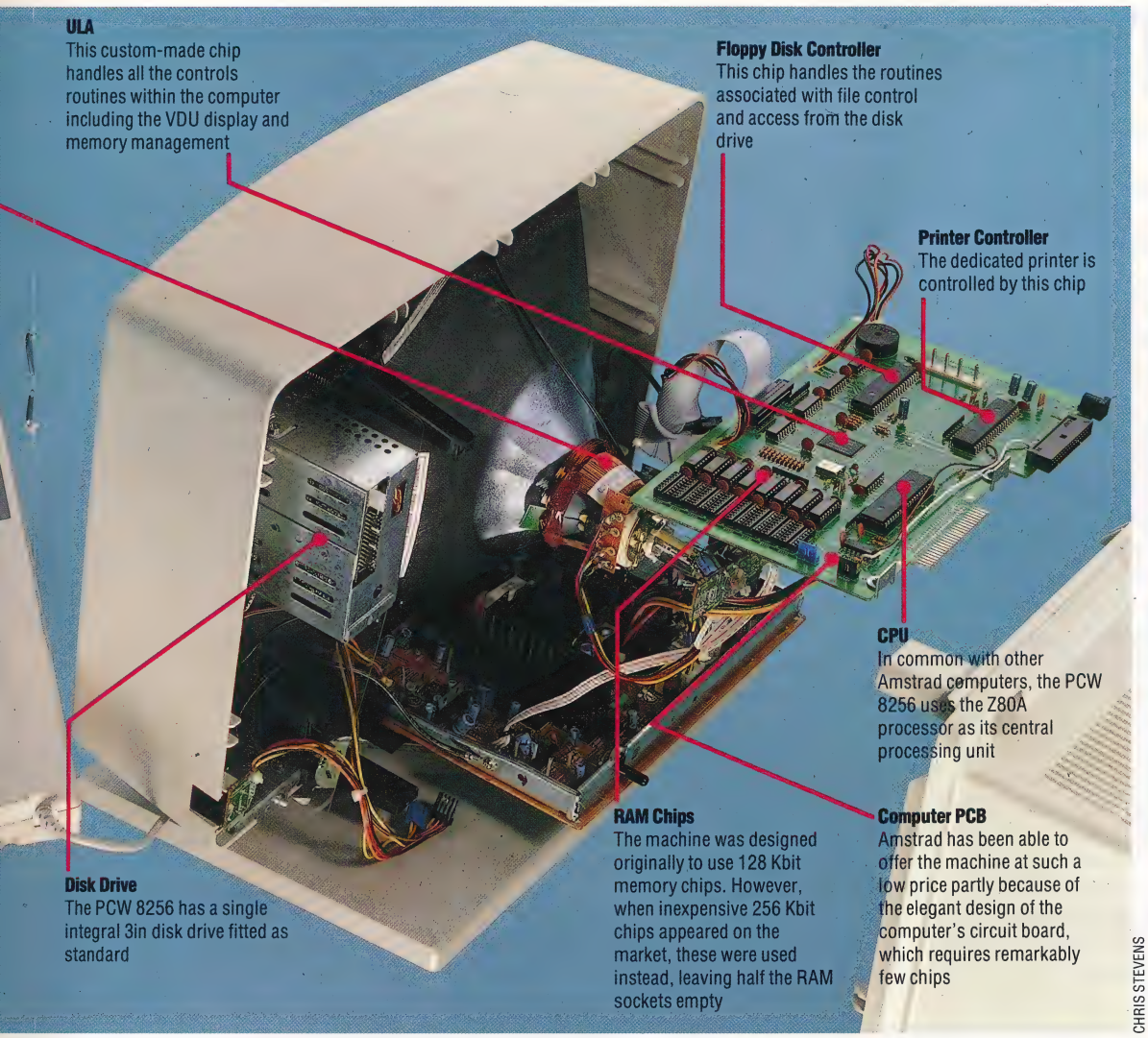
Monitor
The machine's green screen display is provided by this monitor with the signal being decoded by the printed circuit board underneath

Keyboard
Like many other dedicated word processors, the PCW 8256 contains a number of additional keys not found on standard computers, which are used specifically for word processing functions





Plum Loco
LocoScript, the custom-designed word processing software for the PCW 8256, supports a wide range of features. While many facilities are available from a series of pull-down windows at the top of the screen, many others are embedded in the text. Although it is useful to examine these commands on-screen, many users might feel that they clutter the text while writing. However, it is possible to remove them from the screen display


ULA
This custom-made chip handles all the controls routines within the computer including the VDU display and memory management
Floppy Disk Controller
This chip handles the routines associated with file control and access from the disk drive
Printer Controller
The dedicated printer is controlled by this chip
CPU
In common with other Amstrad computers, the PCW 8256 uses the Z80A processor as its central processing unit
Computer PCB
Amstrad has been able to offer the machine at such a low price partly because of the elegant design of the computer's circuit board, which requires remarkably few chips
RAM Chips
The machine was designed originally to use 128 Kbit memory chips. However, when inexpensive 256 Kbit chips appeared on the market, these were used instead, leaving half the RAM sockets empty
Disk Drive
The PCW 8256 has a single integral 3in disk drive fitted as standard

CHRIS STEVENS

AMSTRAD PCW 8256

PRICE
£460 inc VAT

DIMENSIONS
375×326×309mm

MEMORY
256K RAM, of which 128K is configured as a RAM disk

CPU
Z80

VIDEO DISPLAY
92 × 32 characters;
24 lines × 90 characters, text area in LocoScript

INTERFACES
Second disk drive socket, printer interface, keyboard connector

SOFTWARE PROVIDED
CP/M version 3.0, LocoScript word processing program, Mallard BASIC, Dr LOGO, GSX graphics utility

DOCUMENTATION
The manual is comprehensive, as we have come to expect for Amstrad computers

STRENGTHS
At the price, the PCW 8256 is a bargain. For around £460 the user receives a 256K computer, monitor disk drive and printer. At present there is no other offer on the market to compare

WEAKNESSES
Some of the hardware shows signs of corner cutting. In particular, the printer appears to be slow and of dubious reliability. Furthermore, it appears that even with a special interface a different printer would be unable to take advantage of many of LocoScript's features

windows as a means of simplifying the system. Although you no longer have to learn the numerous Control characters used on most word processors, each of the function keys on the Amstrad machine leads to its own menu, and some of the labels of both keys and menus are unclear as to their purpose. This means that you'll still have to learn a number of paths and keypresses in order to achieve the desired effect. (Note that although the PCW 8256 does not have a Control key, the Alt key is substituted for many CP/M-based programs.)

COMMAND ENTRY

LocoScript does provide a quicker method of entering commands. On either side of the Space bar are keys marked ± and =. In order to right-justify a line, pressing the ± key followed by RJ will execute the command, while = will terminate it. This still takes more time than using a single control character — a toggle mechanism would have been simpler.

Adding to the confusion is some of the on-screen editing. Once a paragraph has been altered, some of the lines may look ragged because the software has not justified the paragraph, forcing the user to do so by pressing the Relay key. This process of manual justification is common in older word processors, in particular WordStar, but one might have expected a package developed in 1985 to have incorporated automatic justification.

Another curious feature of LocoScript is that it appears as though the computer's operating system is incapable of reading many of the files. This is as a result of CP/M 3.0 being able to support a number of different users. When the operating system is loaded, it automatically defaults to user 0. By searching through the files listed under different user numbers, the operator will come across the 'missing' LocoScript documents.

Despite these problems, the PCW 8256 is still great value for money and contains a number of excellent features. The range and variety of the printer and page formatting commands are similar to those available on packages bundled with micros costing several times the price of the Amstrad machine. It seems that this time around, however, Amstrad's policy of offering the maximum number of features for the minimum cost has been over-stretched. Users may feel that a little better design — in particular the printer — even at the cost of some of the utilities, would have made the PCW 8256 a much better proposition.

FOREST RANGER

Hierarchical directories are a convenient and useful method of dealing with large numbers of files. We look at the MS-DOS commands that allow disk directories to be manipulated and consider other utilities.

All the resident commands that we looked at in the previous instalment are available in all versions of MS-DOS. From version 2.0 onwards, however, a new range of features became available — most of them borrowed from Unix. Of prime importance was the introduction of a *hierarchical directory* structure. This is of particular value with modern high-density floppy disks, and absolutely essential for hard disk systems.

When a DOS 2 system is first booted, the user 'sees' an area of the default drive containing a number of files. In most cases these will be either programs (.EXE or .COM) or data files. It is usual to have as much as 720 Kbytes of storage on a modern 3½in microfloppy, and several tens of megabytes on a hard disk. The large number of files means that a simple directory listing might entail a huge amount of information flashing by — most of it irrelevant to the user's current area of interest. A method of splitting up the disk area into manageable 'sub-directories' is clearly required.

The default directory is called the 'root' and a sub-directory may be created within the root with the command md (short for mkdir, or MaKe DIRectory). Thus:

```
md work
```

would create a 'file' (called work) on the root directory that 'contained' other files rather than straight data.

We might wish to create some documents or program source text within this new directory, so we CHange DIRectory by giving the command chdir, or its alternative:

```
cd work
```

If we now type dir (to list the contents of our 'current directory'), we might expect it to be empty. MS-DOS has already created two sub-directories within work, however. They are named . and .. (somewhat cryptically) and denote the current directory (work) and its 'parent', respectively. Thus, if we entered the command:

```
dir ..
```

a listing would be given of all the files in the root directory. This would now include the entry:

```
WORK    <DIR>    17-11-85    11:21a
```

If we now gave the commands:

```
mkdir today
chdir today
```

we would find ourselves 'in' a sub-directory that had work as its parent directory, and the root as its 'grandparent'.

When specifying the name of any file on the system, we must give a full 'path name' if the file is not in the current directory. Thus, for example, a program on the root directory could be specified as either ..\..\prog or simply \prog (the backslash being used as a separator in path names or, if the first symbol in a path name, as an abbreviation for the root). Moving 'up' one level (to the sub-directory work) can be accomplished with either chdir .. or, in this case, cd \work. If we now entered md source, a 'sister' directory to today would be created, both of them being regarded as 'daughters' of our current directory work.

As you can see, this structure begins to resemble a family tree, and is in fact called a 'tree' of directories. Adding a few more branches might produce the structure on the following page.
A full path name for a file called MEMO.DOC in the directory text would therefore be:

```
\work\source\text\memo.doc
```

This would locate the file from anywhere in the tree, as a full path name is specified starting from the root (\); if we were currently in the sub-directory today, however, we could use:

```
..\source\text\memo.doc
```

At any time a dir command will give a listing of just the current directory, but an alternative path may be specified, as in:

```
dir b:\pascal\source\newprogs
```

Notice that the final name in this case may be that of a directory, not a file, and that a different disk drive may be given as the first item in the path (in this case drive b). As long as a disk, is not removed from a drive, any sub-directory currently selected on that drive will be 'remembered' by the disk. This means that:

```
copy \*.* b:
```

will copy all files on the root directory of the current disk to whatever directory is current on drive b. Each disk, or 'volume', may be given a name — either when originally formatting the disk or by means of the transient utility LABEL.EXE.

TRANSIENT UTILITIES

LABEL.EXE and FORMAT.EXE are two of a number of programs that are supplied as 'transient' utilities. These are separate programs (usually having the extension .EXE, but sometimes .COM) that can be executed by entering their primary names. In order to format a disk, for example, the program FORMAT.EXE can be run by entering:

```
format b:
```

The FORMAT command takes two optional

'switches' delimited by a forward slash. The options are:

```
\s To copy the MS-DOS system after formatting.
\p To ask for a volume name or 'label' to be given to
   the disk. Only one (root) file that has this attribute
   is allowed.
```

This assumes that the current working directory contains FORMAT.EXE — but supposing it doesn't? DOS 2 provides an extremely useful resident command to enable the definition of default path names:

```
path=A:\system;A:\system\utils
```

Entering an unrecognised command will cause the system to follow the path (or series of alternative paths separated by semicolons) in order to find the command. You can find out what the current search path is by entering:

```
path
```

and if none has been defined, the message No path will be printed.

EXTERNAL COMMANDS

Many versions of MS-DOS will be supplied with external commands appropriate to the OEM system, but there are some that are so useful that they will be found in all versions. PRINT.EXE enables 'background' printing of a text file while the user continues to work at the computer. EDLIN.EXE is a line editor that enables the creation of text files without the need for an additional word processor. DISKCOPY.COM (or DCOPY) copies the entire contents of a disk, and DISKCOMP (or COMP) may be used to verify an uncorrupted transfer.

All MS-DOS utilities involving the conceptual use of two disk drives are sufficiently 'intelligent' to realise the need for disk swapping when running in a single-disk environment. For example, DISKCOPY loads data from a source disk into memory and then issues a prompt to change disks. When that copy operation is completed, another prompt to insert the source disk is given — and so on, until the whole disk has been transferred. Bad sectors are detected, and the destination disk may be formatted at the same time as copying.

The most useful utility for checking disks is CHKDSK.EXE, which gives complete information about the state of a disk, including flagging the existence of 'hidden files'. Used without arguments, the command:

```
chkdsk
```

checks the currently 'logged' drive and produces a report in the following form at:

```
720666 bytes total disk space
 47528 bytes in two hidden files
  2048 bytes in four directories
526386 bytes in 39 user files
144704 bytes available on disk

262144 bytes total memory
198726 bytes free
```

Usefully, CHKDSK reports on the system memory status (the last two items) and also checks general disk usage and the File Allocation Table (FAT) held for each device. MS-DOS can sense when media are changed and rebuild a FAT for a new disk by reading system information from it.

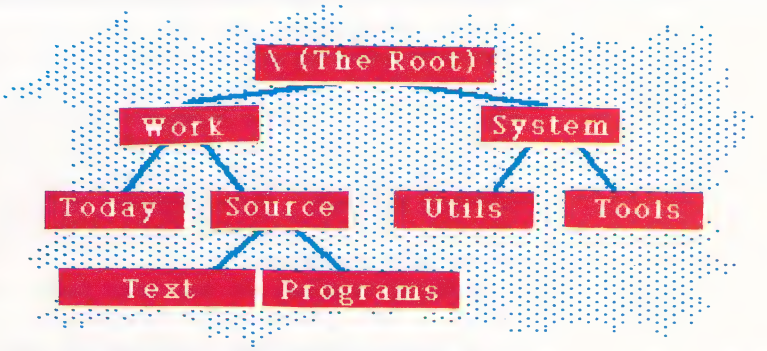
One word of warning: never use the DOS CHKDSK utility supplied with one version of MS-DOS to check DOS disks of another major version. The disk sectors and directory information can be arranged in a quite different format, and this can be particularly dangerous if DOS 1.x and DOS 2.x are 'live' on the same site. The resulting mess of intertwined links to the wrong disk sectors could mean hours of fruitless work trying to recover, say, a document that had 'pages' of machine code from another file interleaved with it. If you mistakenly use CHKDSK from DOS 1 on a DOS 2 hard disk system with 10 or 20 Mbytes at risk, the consequences can be disastrous.

If you have cause to upgrade an old IBM PC or Sanyo from DOS 1 to DOS 2, the moral is obvious: back up all existing data on formatted - non-systemised disks, and then destroy all current systems (by reformatting if necessary) before installing DOS 2. OEMs supplying the better implementations of MS-DOS ensure that transient utilities incorporate code to check that the system they are executing on is a compatible version, but you still need to be careful.

In the next instalment of this series we will take a closer look at the two mysterious hidden files normally present on a system disk, examine the structure of the operating system itself and explore the power of I/O redirection and 'pipelines'.

Division Of Labour

The hierarchical file directory system implemented in MS-DOS enables users of large capacity disk systems to divide the files into subcategories, since directories can not only contain filenames but other directory names. In the example directory structure shown here, the file 'text' is a member of the directory 'source', which is itself a member of the directory 'work'. The root directory contains the main categories of file groups within the tree structure



CAROLINE CLAYTON

CLASS CONSCIOUS

Our analysis of the C language moves on to consider variables in terms of their 'storage class', and the way in which arrays are initialised. Variables are recognised as having a distinct status and are assigned memory in different ways according to which of the four classes they belong.

As well as being of a particular type, all C variables have a *storage class*, which determines the way in which the compiler recognises them and assigns memory for them. There are four classes: automatic, external, register and static. Variables can have their class specified by placing the key words auto, extern, register or static in front of the type in the declaration. For example:

```
extern double x, y;
```

Automatic is the default class for nearly all variables. Any variable declared within a function body will be automatic by default and local to that function. Every time the function is entered, therefore, new storage space will be allocated for that variable and this will be lost upon exit from the function. There is no way that a value can be retained by such a variable between two calls to the function. The same applies to any variable defined within a block of code, indicated by being enclosed in braces {}. Remember that main() is a function, so even variables declared here will normally be automatic.

External variables, on the other hand, are global — that is, they can be referred to at any point within the program, and even in certain instances by functions that appear in a different file of source code from the declaration. A variable declared outside a function body will be considered external by default. An external variable can also be declared at any other point within a function body or block. It will remain in existence even after a function finishes execution and can be referred to at any point in the program following the declaration.

If external variables of the same name are declared in two or more files of source code they will be considered as the same variable when the files are linked together. If a local variable is declared with the same name as a global variable it will 'mask' the global variable within its scope, so references to that variable name will be to the local variable within its scope and to the global variable outside.

Register variables behave exactly like automatic variables, and indeed in some cases will be treated the same. However, if it is possible the compiler will allocate space in high-speed storage for them. This is clearly going to depend very much on the processor being used to run the program, and on the size and number of its registers in particular. A 68000, for example, with its 16 general-purpose 32-bit registers, has reasonable scope for using the registers for a limited number of register variables, whereas a Z80 would probably have no space. Register variables should be used only sparingly and be declared as late as possible to restrict their scope and free the space as soon as possible. Loop control variables are common candidates for this class.

Static variables are usually local to a function or block, but they differ from automatic variables in that the storage allocated and the value stored therein is retained between executions of the function or block. One possible use of a static variable might be to count the number of times a function is called. They can also provide 'data hiding' in that the values stored in such variables are not accessible externally, unlike external variables. A static variable that is declared externally to a suite of functions is globally available to those functions but not available to functions outside that particular file.

THE USE OF ARRAYS

An array is declared just like other variables with its size (that is, the number of elements), in square brackets following the variable name. For example:

```
int intarray[100]:
```

This reserves storage space for array elements intarray[0], intarray[1], and so on up to intarray[99]. Subscripts in C always start at zero, and the declaration refers to the total number of elements, so in this case there is no element intarray[100]. Static or external arrays can be initialised by adding a list of values enclosed in braces to the declaration. For example:

```
static int days_in_month[12] =
        {31,28,31,30,31,30,31,31,30,31,30,31};
```

If the list is incomplete, the remaining elements will be set to zero. If no initialisation is done for a static or external array then all elements will be initialised to zero. Automatic arrays cannot be initialised and the space will be created with garbage values, so you cannot rely on elements being initialised to zero. The idea of a register array obviously does not make sense.

If an array is being initialised then C does not require the size to be mentioned — this will automatically be taken as the number of values provided, so the above declaration could equally well have been written as:

```
static int days_in_month[] =
        {31,28,31,30,31,30,31,31,30,31,30,31};
```

This facility is particularly useful for strings, or arrays of type char, where the initialising string can

simply be enclosed in quotes. So the following two declarations are equivalent:

```
static char st[] = "hello";
```

and

```
static char st[] = {'h','e','l','l','o'};
```

Note, however, that these strings are not dynamic in the same sense as BASIC strings — their length cannot be varied from that declared. We will look more closely at string handling in a later instalment.

C can cope with arrays with virtually any number of dimensions. The only point to remember with arrays of two or more dimensions is that each subscript must have its own pair of square brackets, so the declaration for a two-dimension array of four by five elements would be entered as:

```
int twodarray[4][5];
```

Arrays can be passed as parameters to functions, but in this case they are being passed by reference — that is, the address of the first element is passed to the function and any change in the array effected within the function will remain in effect when control is transferred back to the calling routine. It is not necessary to declare the size of an array within the function since this is known anyway, so functions can be written quite generally to operate on arrays of any size.

The example listings we give uses many of the concepts discussed here. We are assuming that a random number generator is required and the functions to manipulate it are written in one file, which can be linked in to any program needing random numbers. In a separate file is a short test program that calls for a large set of random numbers and counts the frequencies of numbers in given ranges to check for a uniform distribution.

Listing 1, stored in file 1

```
#define MULT 25173
#define MODULUS 65536
#define INCREMENT 13849
static int seed;

/* this declaration is external as far as this file is
concerned so the variable seed is available to all
functions in this file, but it cannot be referred to
outside the file; indeed the name 'seed' can be
used freely in other files */
randomise(s)

int s
{
    seed=s;
}
random(n)
int n

/* returns a random number between 0 and n */

{
    seed = (MULT * seed + INCREMENT) %
    MODULUS;
/* the linear congruential method */
    return((int) ((double) seed / (double)
    MODULUS *
    (double) n + 0.5));

/* note the use of casts to convert the integer
    values to floating point for the arithmetic and
    back to int, the + 0.5 is to round to the nearest
    integer */

}
double rnd()

/* returns a random number between 0 and 1 */

{
    seed = (MULT * seed + INCREMENT) %
    MODULUS:
    return ((double) seed / (double) MODULUS);
}
```

Listing 2, stored in file 2

```
#define SEED 17
#define SIZE 100
#define LIMIT 10000
#define NUM__OF__GROUPS 10
int groups [] = {10,20,30,40,50,60,70,80,90,100};

/* external array can be initialised; this one
contains group boundaries for counting
frequencies */

int frequencies [10];
/* this external array will be initialised to zeros */
main()
int r;
{
        randomise(SEED);
        register int i;

/* using register class for loop variables for most
efficient processing, and saving declaration as
late as possible */

        for (i = 0; i < LIMIT; ++i)
        {
/* get 10000 random numbers between 0 and 100 */
                r = random(SIZE);
                register int j;
/* check to see which group they belong to */
                for (j = 0; j < NUM__OF__GROUPS;
                        ++j)
                {
                        if (r < groups[j])
                        {
/* count the random number in the appropriate
group and break out of the loop */
                                ++frequencies[j];
                                break;
                        }
                }
        }
/* print out a frequency table to check for uniform
distribution */
        for (j = 0; j < NUM__OF__GROUPS; ++j)
        { printf("%d—%d = %d\n",groups[j]—
10,groups[j],frequencies[j]) }
}
```

COMMAND PERFORMANCE

In the last instalment, we discussed the multiply and divide operations, and considered the BCD add and subtract instructions. We will now move on to look at the 68000's logical instructions, the shift and rotate operations and the all important program control or branch instructions.

Let's first consider the logical instructions available on the 68000. The AND instruction ANDs the source with the destination, placing the result into the destination and setting the N and Z bits in the status register appropriately. There are a large number of addressing modes allowed, but the source or destination must at least be a data register. Moreover, the data attributes are allowed; so, for example, if D0 = 1010 1010 and D1 = 1111 0000, and we execute the instruction:

```
AND D1, D0
```

then D0 becomes D0 = 1010 0000.

This is a very common example of the use of an AND instruction. In this case, we have marked out the least significant four bits by setting the mask bits in D1 to a 0, and have allowed whatever state the MSBs are in to appear in the destination by setting the mask bits to a 1. Perhaps a more explicit version of this would include the ANDI instruction. This allows the immediate mode of addressing as a source operand, and data-alterable modes as a destination. Using ANDI in the previous example, we would have:

```
ANDI #$F0,D0
```

The remaining logical instructions are OR and ORI for logical OR, EOR and EORI for exclusive OR, and NOT. These instructions follow much the same pattern for allowed addressing as the AND instruction and the condition code settings are the same. Bit manipulation with the logical instructions is very important for control of 'flag' bits in, for example, a word or the control of digital input/output lines for peripherals or external equipment. We'll discuss this later when we examine I/O.

Where the data operand is a single bit, the 68000 has a set of four bit manipulation instructions that can be used instead of the logical instructions. These instructions test the state of a specified bit — numbered 0 to 31, in a data register (long word only) — or a byte in memory. They also set the Z bit in the SR according to the state of that bit. Hence the Z bit becomes an inverted one-bit memory of the specified bit. The instruction BTST can be used as follows. If, for example, D0 = XXXX XXXX XXX1 0000 (binary, where X is 0 or 1), then:

```
BTST #4,D0
```

would test bit four and set Z to zero (the bit in question is not zero). Additionally:

```
BTST #3,D0
```

would set Z to one. All other condition codes would be unaffected.

The remaining instructions in the set affect the bit being tested. They are:

```
BSET    bit test and set
BCLR    bit test and clear
BCHG    bit test and change
```

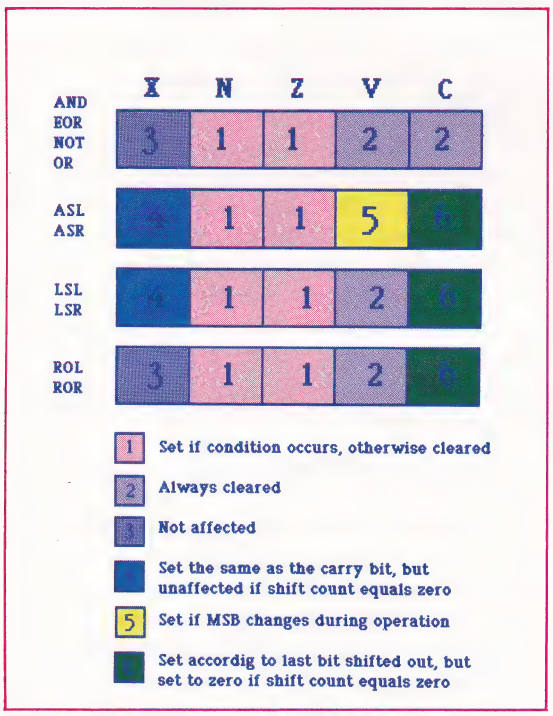
So, if we executed:

```
BCHG    #4,D0
```

in the previous example (where D0 was set to 1, then D0 would become D0 = XXXX XXXX XXX0 0000 and Z is set to 0, indicating the state before the change.

Notice that all these instructions carry out the test and bit setting operations in one instruction. This could be important in a multi-programming environment where the possibility of being interrupted between these two operations could lead to unpredictable results. A final point on logical instructions is that even though the test has been applied, you don't have to take any resultant action. You can just use the instruction as bit manipulators.

Status Update
The logical, shift and rotate instructions affect the contents of the status register in the manner shown here. The arithmetic shift instructions differ from the logical shifts only in their use of the V flag to indicate possible changes in the sign bit of the operand


X N Z V C
AND EOR NOT OR
3 1 1 2 2
ASL ASR
4 1 1 5 6
LSL LSR
4 1 1 2 6
ROL ROR
3 1 1 2 6
1 Set if condition occurs, otherwise cleared
2 Always cleared
3 Not affected
Set the same as the carry bit, but unaffected if shift count equals zero
5 Set if MSB changes during operation
Set accordig to last bit shifted out, but set to zero if shift count equals zero

THE SHIFT OPERATION

First of all, let's look at what happens when a bit pattern is shifted left or right. If D0 holds 0000 0000 0000 1000, then by shifting this pattern three places to the right we get 0000 0000 0000 0001. The contents of D0 have been divided by eight (or 2^3),

so in other words a right shift corresponds to division by two for each place we shift right. Furthermore, it follows that a shift left will result in a multiplication by a similar amount.

Notice that in our example, we have assumed that zero is brought in as the new binary digit (from the left for a right shift and vice versa). This would have to be changed if we wanted to preserve the sign of our number when shifting to the right. So, for example, if D0 = 1111 1111 1111 0000 (—16 decimal) then a right shift of three would produce D0 = 1111 1111 1111 1110, which is —2 in decimal. Here, we have inserted ones from the right hand in order to maintain the negative number. In general, we say that for right shifts the sign is maintained by bringing in bits of the same sign as the sign bit (the most significant bit of the data operand). For left shifts, we always bring in zero into the least significant bits of the word. If we wish to maintain the sign of the operand in this manner then the shifting operation is called an *arithmetic shift.*

On the 68000 these arithmetic shift instructions are ASL, for arithmetic shift left, and ASR, for arithmetic shift right. If the destination is a data register then we can shift by up to eight bits using immediate mode for the number of shifts, or use the contents of another data register as the shift count and shift up to 31 bits of a long word. So, for example:

```
ASR     #8,D0
```

would be the maximum shift for immediate mode, and:

```
ASL     D1,D0
```

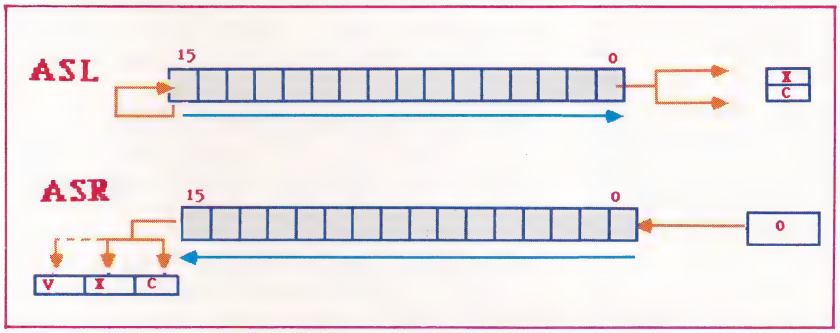
would allow us to shift D0 left by the number of places contained in D1 (that is, up to 31 places). If, however, we wanted to shift a memory location, then the only option is to shift one place at a time. For example, ASR FRED would shift the location given by FRED one place to the right .

You will notice from the status control diagram that all the condition codes in the SR are affected — the C and X bits reflecting the shifted out bits, and the V bit indicating any change of the sign bit for left shifts. The other bits, N and Z, are set according to the result.

You may wonder why these arithmetic shifts are considered so important, especially as the 68000 provides multiply and divide instructions. The reason lies in the execution time the two classes of instruction take. A MULT operation takes 70 clock cycles, but an ASL or ASR takes 6+2n cycles, where 'n' is the number of shifts performed. Thus, an arithmetic shift execution time ranges between eight cycles for a one shift, to 68 for a full 31-bit shift. In this way, for a small number of shifts, the shift instructions will be considerably more efficient than the multiply or divide instructions. In the extreme case of a divide by two the gain will be 19 times quicker!

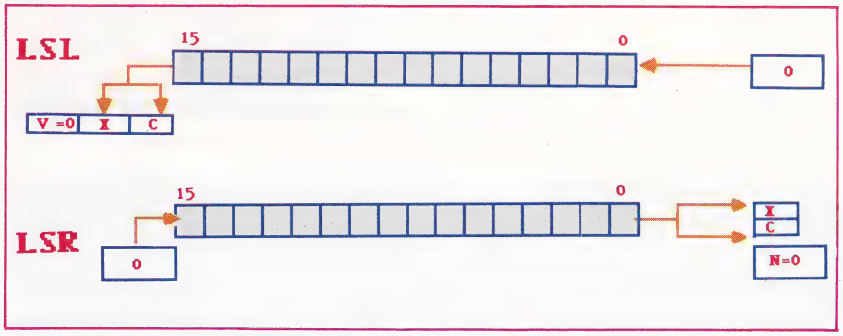
Note that we can also have *logical shift* instructions, LSL and LSR, which do not maintain the sign of the data operand and always bring in zeros as the new binary bits. The addressing restrictions of the arithmetic shifts apply to the logical shifts too, and the V bit is always set to zero. Logical shift can usefully be used when setting up or testing data operands containing smaller sub-groups or fields.

Finally, in this set of shift instructions, we come to the rotate group of instructions. These are very much a hang-over from the past, because their main use is in testing individual bits of data operands by rotating and testing the condition flag settings that result. On the 68000, of course, we have a comprehensive set of bit testing and setting instructions and these take about the same execution time as the rotate set. So perhaps a simple example of rotation will be sufficient here. ROR #3,D0 will rotate the contents of D0 three to

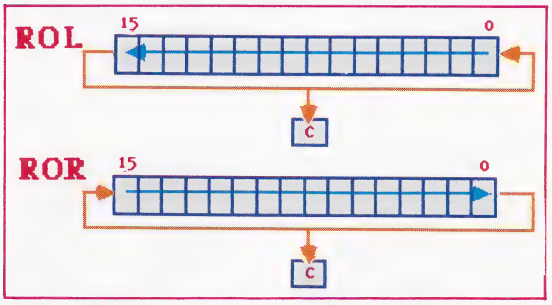


Arithmetic Shift
When shifting bits to the right (ASR), the arithmetic operation copies bit 15 into bit 14, but the contents of bit 15 remain unchanged. The sign bit is therefore preserved. Bit 0 is copied into both the C and X bits of the status register. Note than when shifting to the left, zeros are copied into bit 0 and the V bit will be set if the contents of bit 15 are changed

Logical Shift
The logical shift operations simply move zeros into either bit 15 or bit 0, depending on whether a left or right shift is involved. Note, however, that V is set to zero during LSL, and N is set to zero during LSR. The last bit to be shifted out is stored in C and X in both operations



CAROLINE CLAYTON



Rotate Instructions
The ROTATE instructions simply rotate the operands, moving bit 0 into bit 15 during ROR and vice versa for ROL. Note that the 'swap-over' bit is copied into the C bit of the status register during the operation

the right, and set the C bit according to the last bit shifted round, least significant to most significant.

PROGRAM CONTROL INSTRUCTIONS

Let's now move on to look at the program control instructions. These are a most important set of instructions because they control the instruction execution sequence. One group, called 'conditional branches', will alter the normal sequential instruction flow depending on a certain tested condition. 'Unconditional branches', on the other hand, will always cause a branch or change in the normal sequential instruction flow. We'll look at these first.

The usual way of executing an unconditional branch is:

```
BRA   NEWLABEL
```

where the instruction flow will continue from the location called NEWLABEL after the execution of the BRA (or branch always) instruction. If the byte displacement can be held in a signed eight-bit word then the resulting instruction will be coded into one word. Half the word will contain the BRA op-code (60 hex) and the other byte will contain the signed eight-bit displacement. If the displacement cannot be held like this, or the branch address is not yet known — that is, it can't be calculated by the assembler because a forward reference is involved — then the displacement byte will contain zero and the next word will contain the full 16-bit signed displacement. We'll see an example of this later.

Normally, the BRA instruction will be adequate; however, on some occasions we will wish to do something more sophisticated when the branch address is computed. For example, when you want to branch to an address contained in a table with an index set in a register (you will remember that this form of addressing we called 'indirect with index and displacement'). Unfortunately, the BRA instruction does not allow this form of computed address, so Motorola has provided the JMP (or 'jump') instruction where computation is permitted for the branch address if required.

Let's look at a program example that shows all these different forms of unconditional branches. The Branching Out listing illustrates the principles involved.

The jump instructions in this example show absolute addressing (forwards and backwards) and indirect with indexed and displacement. Other addressing modes allowed are 'simple indirect' — for example, (A2) — and 'PC relative'.

The BRA instructions show first an example where the address displacement is contained in the instruction word, and then two examples where full extension words have been used. In the first of these (BRA FINISH) a full word extension has been used even though the displacement could have been held in a byte. This is because the assembler does not know the address of FINISH yet, and therefore must allow for a full word extension for the displacement. If you know that the forward displacement will fit in a signed byte then you can force the assembler to use the short form of the instruction by using a .S suffix. So, our example could more efficiently be written as BRA.S FINISH.

CONDITIONAL BRANCHES

We'll consider now the second set of branch instructions, called the 'conditional branches'. This set is further subdivided into three subsets:

- Two's complement branches
- Unsigned branches
- Loop control

The first two of these subsets have a common instruction format of:

```
Bcc LABEL
```

in which the cc refers to the condition code being tested. If this condition is true then a branch to LABEL takes place; otherwise the next sequential instruction is executed. The conditions tested are shown in the Branch Conditions table.

The 'True If' column in the table is the arithmetic condition that applies as a result of the comparison using a CMP or a SUB instruction (which should obviously be executed immediately prior to the conditional branch being applied). In the case of the first subset of conditions shown in the table (the two's complement branches), the V bit, or overflow flag, is included in every case in the logical testing and this is the factor that determines membership of this subset. It also implies that additional testing of the overflow bit is required to complete correctness, and careful examination of the logical conditions for a branch should be made in the *Motorola User Manual*. For example, a BGE branch tests to see whether N=V, and a branch will be made if NOT N AND NOT V is true (that is, if there is no overflow and it's not negative), or when both N and V are true (overflow and negative).

Let's look at an example. Suppose we wish to

Branching Out

```
                    WAYOFF  EQU     $3FFF
                            ORG     $1000
100     3200        START   MOVE    D0,D1
1002    4EF8 1000           JMP     START       *Full word extension
1006    4EF8 101C           JMP     FINISH      *And again but forwards
100A    4EE8 0005           JMP     5(A0)       *Indirect with displacement
100E    4EF2 000C           JMP     12(A2,D0)   *Indirect with index and
                                                 displacement

   *Branch always versions

   *Displacement in op-code word if contained in a byte

1012    60EC                BRA     START       *Negative displacement
1014    6000 0004           BRA     FINISH      *Forwards, not known yet
                                                 how far
1018    6000 2FE5           BRA     WAYOFF      *Gives full 16-byte
                                                 displacement word
101C    3200        FINISH  MOVE    D0,D1
```

compare the two signed numbers in D1 and D2 and branch to BIGGER if D2 is greater than D1. If the condition is tested first with a compare instruction, then the BGT conditional branch is used, as follows:

```
CMP   D1,D2        *forms D2-D1
BGT   D2BIGGER     *branch to D2BIGGER if not
                   *zero and not negative and
                   no
                    overflow
```

The second subset of conditional branches shown in the table involves unsigned numbers and relatively simpler testing of condition codes. For example, comparing the contents of a location BETTY with D1, irrespective of any overflow condition, could be performed by the sequence:

```
CMP   BETTY,D1     *form D1 — BETTY
BEQ   EQUAL        *branch to equal if z = 1
```

Another example of these conditional branches is when the assembler equivalent of a loop is coded. For example, the assembler equivalent of:

```
FOR I := 1 to 5 do
(. . . program body to be executed 5 times . . .)
NEXT I
```

is:

```
      MOVEQ  #5,D7    *set loop count
LOOP (. . . program body to be executed 5 times . . .)
      SUBQ   #1,D7    *decrement count
      BNE    LOOP     *repeat until D7 = 0
```

which is efficiently coded into a mere three words (because 'quick' instructions have been used).

The third subset of conditional branches is a single instruction, DBcc, meaning decrement and branch on condition cc. This instruction is an extension of the loop control program given above, but with the decrement and conditional branch all coded into one instruction. In fact the instruction follows the PASCAL-type pseudo-code for a repeat loop:

```
REPEAT
   (. . . body of loop . . .)
UNTIL
   'c' = true or Dn = −1
```

where cc is one of the conditions described in the second subset shown in the table and Dn is a data register used to hold a loop count. Let's look at how this would be coded with the DBcc instruction:

```
      MOVEQ  #5,D1    *set loop count
LOOP (. . . body of program . . .)
      DEBQ   D1,LOOP *exit if last tested 'cc' is 0
                      *or D1 = 1 (6 iterations)
```

Notice carefully the exit conditions explained in the comments above, and how the branch sense is different to a normal BEQ instruction. If the conditional testing is not required then a cc of F (for false) may be used to give the DBcc equivalent of a simple FOR loop. For example:

```
      MOVEQ  #4,D3
LOOP (. . . body of FOR loop . . .)
      DBF    D3,LOOP
```

This would be the same as our original five-iteration FOR loop, and still occupies the same amount of memory space (all DBcc instructions take two words). Some programmers might feel, however, that our original code was more explicit.

The 68000 provides a wide variety of conditional and unconditional branch instructions, which can be very powerful when used properly, particularly with careful consideration of the condition code testing.

Branch Conditions

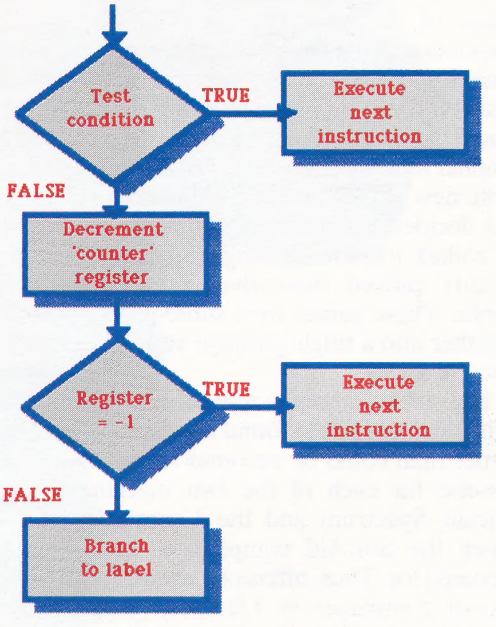
The following conditions may be tested before a branch to an address calculated using two's complement arithmetic

cc	Condition	True If
GT	greater than	destination > source
LT	less than	destination < source
GE	greater than or equal to	destination >= source
LE	less than or equal to	destination <= source
VS	overflow	V = 1
VC	no overflow	V = 0

The following conditions may be tested before a branch to an address calculated using unsigned integers

EQ	equal to zero	Z = 1
NE	not equal to zero	Z = 0
MI	minus	N = 1
PL	plus	N = 0
HI	higher than	C = Z = 0
LS	lower than or same	C or Z = 1
CS	carry set	C = 1
CC	carry clear	C = 0

On One Condition . . .

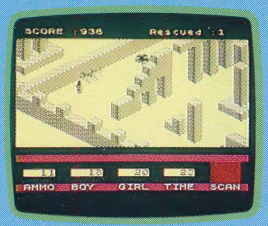


Testing Times

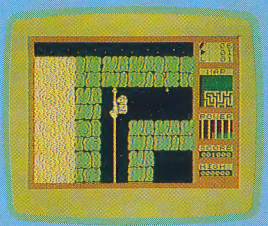
DBcc provides the 68000 programmer with some powerful high-level programming opportunities. The flow chart of the instruction is shown here. Note that it does not simply decrement a counter register (compare this with the Z80 DJNZ instruction), but also tests a condition. Sometimes, however, we will want to execute a body of code a certain number of times regardless (as in the BASIC FOR . . . NEXT loop) and for those purposes the conditional FALSE may be used (DBF)

FEED THE WORLD

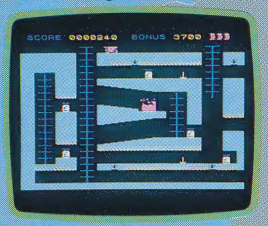
Ant Attack



Fred



Gilligan's Gold



Kokotoni Wilf

Sorcery

SCREEN SHOTS COURTESY OF YOUR SPECTRUM

Charity Show
Shown here is a selection of games from the Spectrum version of SoftAid. The games chosen for inclusion in the compilation were proven arcade-style classics. All royalties and profits generated by sales of the cassette have been donated to the Band-Aid Trust

SoftAid is an organisation formed by a group of software houses who have banded together to raise money to help relieve famine in Africa. By pooling its resources, it has produced a compilation cassette, which has raised thousands of pounds for the Band-Aid charity.

The efforts of the Band-Aid charity, headed by Bob Geldof, undoubtedly captured the imagination of the world in 1985. Following the success of the single Do They Know It's Christmas? at the end of 1984, and the Live Aid concerts in London and Philadelphia in July 1985, a number of organisations in other fields of popular entertainment have given their services free to raise money for the appeal. One of the most successful initiatives came from the microcomputer games software houses, who often like to compare themselves with the pop industry. The result was SoftAid, a series of tapes for the two most popular home micros, containing some of the most successful games on the market.

The driving force behind SoftAid has been the Guild of Software Houses (GOSH), a trade association of the computer games industry whose usual function is to try to stem the tide of software piracy. The Guild asked each of its members if it would like to contribute a game to the project. The original idea was that the software houses would write new games for the compilation, but later it was decided that it was more feasible and realistic to collect together a range of games that had already proved themselves popular with the public. These games were subsequently bundled together into a single package at the special retail price of £4.99.

The software houses responded well, so much so in fact that GOSH found themselves with more games than could be accommodated on a single cassette for each of the two machines — the Sinclair Spectrum and the Commodore 64 — which the SoftAid compilation was originally prepared for. Thus, offers of games from such well known companies as Llamasoft and Software Projects had to be turned down. The games that were selected for the compilation included some of the most popular games of the last few years, including the classic Ant Attack (see page 6).

The full list of the games that went onto the tape for the Spectrum is: Spellbound from Beyond, Starbike by The Edge, Kokotoni Wilf by Elite, The Pyramid by Fantasy, Horace Goes Ski-ing by Melbourne House, Gilligan's Gold by Ocean, Ant Attack by Quicksilva, 3D Tank Duel by Real Time, Jack and the Beanstalk by Thor, and Sorcery by Virgin. For the Commodore 64 the games included are: Gumshoe by A'n'F, Beam Rider by Activision, Star Trader by Bug-Byte, Gyropod by Taskset, China Miner by Interceptor, Kokotoni Wilf by Elite, Gilligan's Gold by Ocean, Fred by Quicksilva, Falcon Patrol by Virgin and Flak by US Gold. During the development of the project, GOSH worked closely with the Band-Aid Trust and as an added bonus, the Band-Aid single Do They Know It's Christmas? was added to the beginning of both of the cassettes, providing yet another incentive to buy the compilation.

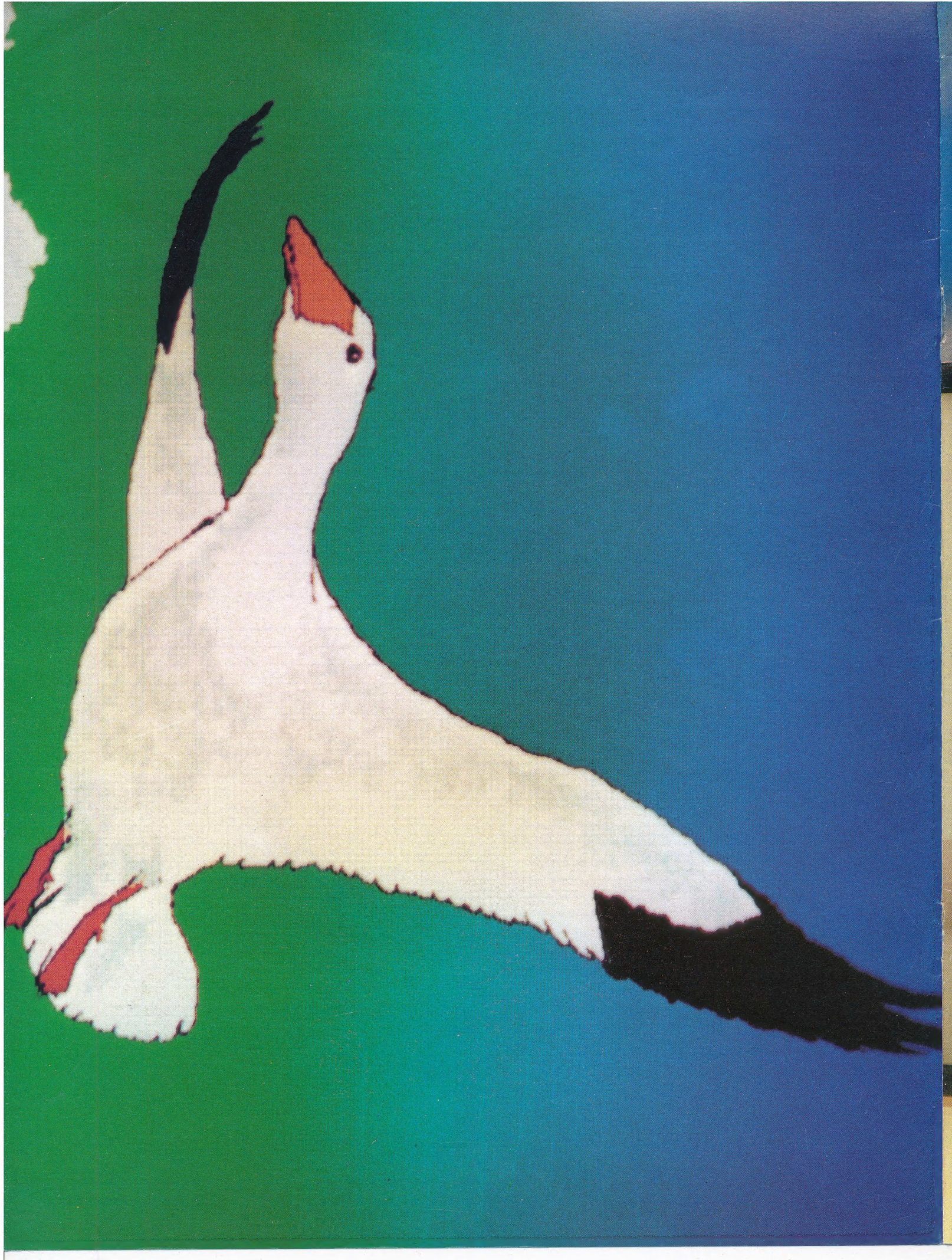
GOSH also observed Band-Aid's general philosophy of asking everybody involved in the production and distribution of the cassette to volunteer their services. Not only were all the games donated free but also the printing, publicity, tape duplication, distribution and retail charges were all waived by the people concerned, with only the cost of materials having to be paid for. The tapes were released in the spring of 1985, immediately went to the top of the best sellers lists for both of the machines involved, and succeeded in raising many thousands of pounds for the Band-Aid Trust. This was despite the fact that many of the customers who bought the tapes quite likely had some of the games already in their collections.

SoftAid: For the Sinclair Spectrum and Commodore 64, £4.99. (Versions for the BBC Micro and Amstrad range are being prepared)
Publishers: The Guild Of Software Houses, 12-13 Henrietta Street, London WC2
Authors: Various
Joysticks: Optional
Format: Cassette

Motorola 68000 Instruction Set

Here, courtesy of Motorola Inc, we present the second of two sheets which give details of the 68000's instruction set

Mnemonic	Description	Operation	Condition Codes				
			X	N	Z	V	C
MOVE from SR	Move from the Status Register	SR → Destination	–	–	–	–	–
MOVE USP	Move User Stack Pointer	USP → An; An → USP	–	–	–	–	–
MOVEA	Move Address	(Source) → Destination	–	–	–	–	–
MOVEM	Move Multiple Registers	Registers → Destination (Source) → Registers	–	–	–	–	–
MOVEP	Move Peripheral Data	(Source) → Destination	–	–	–	–	–
MOVEQ	Move Quick	Immediate Data → Destination	–	*	*	0	0
MULS	Signed Multiply	(Destination)X(Source) → Destination	–	*	*	0	0
MULU	Unsigned Multiply	(Destination)X(Source) → Destination	–	*	*	0	0
NBCD	Negate Decimal with Extend	$0 - (\text{Destination})_{10} - X \rightarrow$ Destination	*	U	*	U	*
NEG	Negate	0 – (Destination) → Destination	*	*	*	*	*
NEGX	Negate with Extend	0 – (Destination) – X → Destination	*	*	*	*	*
NOP	No Operation	–	–	–	–	–	–
NOT	Logical Complement	~(Destination) → Destination	–	*	*	0	0
OR	Inclusive OR Logical	(Destination) v (Source) → Destination	–	*	*	0	0
ORI	Inclusive OR Immediate	(Destination) v Immediate Data → Destination	–	*	*	0	0
ORI to CCR	Inclusive OR Immediate to Condition Codes	(Source) v CCR → CCR	*	*	*	*	*
ORI to SR	Inclusive OR Immediate to Status Register	(Source) v SR → SR	*	*	*	*	*
PEA	Push Effective Address	<ea> → – (SP)	–	–	–	–	–
RESET	Reset External Device	–	–	–	–	–	–
ROL, ROR	Rotate (Without Extend)	(Destination) Rotated by <count> → Destination	–	*	*	0	*
ROXL, ROXR	Rotate with Extend	(Destination) Rotated by <count> → Destination	*	*	*	0	*
RTE	Return from Exception	(SP) + → SR; (SP) + → PC	*	*	*	*	*
RTR	Return and Restore Condition Codes	(SP) + → CC; (SP) + → PC	*	*	*	*	*
RTS	Return from Subroutine	(SP) + → PC	–	–	–	–	–
SBCD	Subtract Decimal with Extend	$(\text{Destination})_{10} - (\text{Source})_{10} - X \rightarrow$ Destination	*	U	*	U	*
Scc	Set According to Condition	If cc then 1's → Destination else 0's → Destination	–	–	–	–	–
STOP	Load Status Register and Stop	Immediate Data → SR; STOP	*	*	*	*	*
SUB	Subtract Binary	(Destination) – (Source) → Destination	*	*	*	*	*
SUBA	Subtract Address	(Destination) – (Source) → Destination	–	–	–	–	–
SUBI	Subtract Immediate	(Destination) – Immediate Data → Destination	*	*	*	*	*
SUBQ	Subtract Quick	(Destination) – Immediate Data → Destination	*	*	*	*	*
SUBX	Subtract with Extend	(Destination) – (Source) – X → Destination	*	*	*	*	*
SWAP	Swap Register Halves	Register [31:16] ↔ Register [15:0]	–	*	*	0	0
TAS	Test and Set an Operand	(Destination) Tested → CC; 1 → [7] OF Destination	–	*	*	0	0
TRAP	Trap	PC → – (SSP); SR → – (SSP); (Vector) → PC	–	–	–	–	–
TRAPV	Trap on Overflow	If V then TRAP	–	–	–	–	–
TST	Test and Operand	(Destination) Tested → CC	–	*	*	0	0
UNLK	Unlink	An → SP; (SP) + → An	–	–	–	–	–

[ ] = bit number
Λ logical AND
V logical OR
⊕ logical exclusive OR
~ logical complement

* affected
– unaffected
0 cleared
1 set
U undefined